Encapsulation

CPSC 233: Introduction to Computer Science for Computer Science Majors II
Winter 2022

Jonathan Hudson, Ph.D.
Instructor
Department of Computer Science
University of Calgary

Wednesday, 10 November 2021

UNIVERSITY OF CALGARY

How do we use information hiding to enforce encapsulation?

Encapsulation

The world sees

The object sees

Encapsulation (Data Hiding)

- Object state:
  - Value of instance variables
  - Should always be valid
- Example:
  - Fraction f = new Fraction(M, N);   // M/N
  - f.denominator = 0;
- Hide your data!
- Declare instance variables private.
  - **Always …. every single instance variable!**

Data hiding is important because you don't want other programmers to modify the data in your classes in unexpected ways.

For example, fractions should never have a denominator of 0. If somebody else is allowed to change f.denominator, then it is difficult to enforce this rule.

Access Modifiers

Specifies who can access what in a class.

- **public** – anyone can access variable/method
- **private** – only code inside the class can access variable/method.
- **protected** – code inside class and sub-classes (inheritance)
- **default** – code in same package (folder)

The default access modifier doesn't use the word "default" in code. These are the access rules we get when we don't use specific access modifiers such as public/private/protected.

Constructors: Instantiating Objects

- Constructors help to enforce encapsulation by defining the allowable ways to create an object.
- Example: Consider a class for Drivers
  - What would be data you would allow to be passed in from “outside” to create an object to store information about new drivers?

Constructors: Instantiating Objects

- Constructors help to enforce encapsulation by defining the allowable ways to create an object.
- Example: Consider a class for Drivers
    - What would be data you would allow to be passed in from “outside” to create an object to store information about new drivers?
    - What would you not allow? (Hint: each jurisdiction should have its own scheme for assigning IDs for driver’s licenses. You shouldn’t be able to create a driver with an arbitrary ID!)

Constructors: Instantiating Objects

- Constructors help to enforce encapsulation by defining the allowable ways to create an object.
- Example: Consider a class for Drivers
  - What would be data you would allow to be passed in from “outside” to create an object to store information about new drivers?
  - What would you not allow? (Hint: each jurisdiction should have its own scheme for assigning IDs for driver’s licenses. You shouldn’t be able to create a driver with an arbitrary ID!)

  - You might have Driver(), Driver (String name), but not Driver(int preferredIDNum)
  - What about Driver(String name, YearMonth dateOfBirth)?

Getting and setting values

Getter methods

- AKA "accessor" method
- Gives the caller access to an instance variable
- Data hiding: only have getters for instance variables others are permitted to access

```
public int getFoo()
{
        return foo;
}
```

Setter methods

- AKA "mutator" method
- Allows the caller to set the value of an instance variable
- Data hiding: only have setters for instance variables others are permitted to change, and enforce rules about what values are permitted

```
public void setFoo(int newFoo)
{
        this.foo = newFoo;
}
```

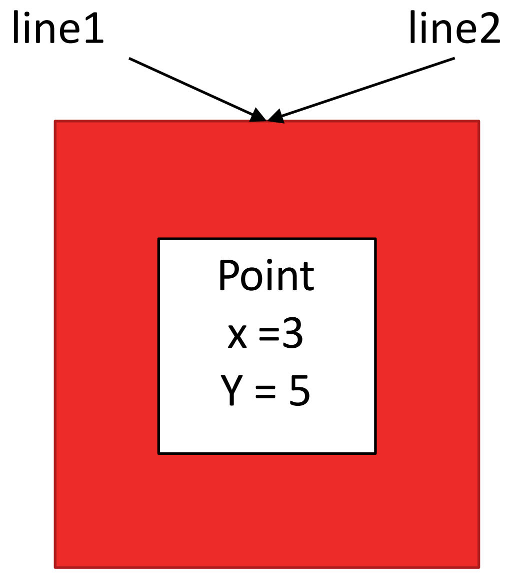
Getting and setting values

- Outside of longer syntax the process of writing getter/setter is one of more disliked properties of Java
  - I have object with 10 fields of data
    - Get all data…10 getters?
    - Change all data…10 setters?
- Other languages now have added syntax that auto-generates these capabilities (with ability to change this behaviour by adding your own code)
  - Note sometimes you don't want data to be accessible via get, sometimes your set is more complicate than x = new value.
- Most IDE's have 'generate' options that will let you pick fields and create set of basic get/set commands as desired
- IntelliJ Code->Generate (Alt+Insert)
  - You will need to have that object field selected at time of attempt

How does Java manage memory, and how does this affect encapsulation?

Data hiding and objects

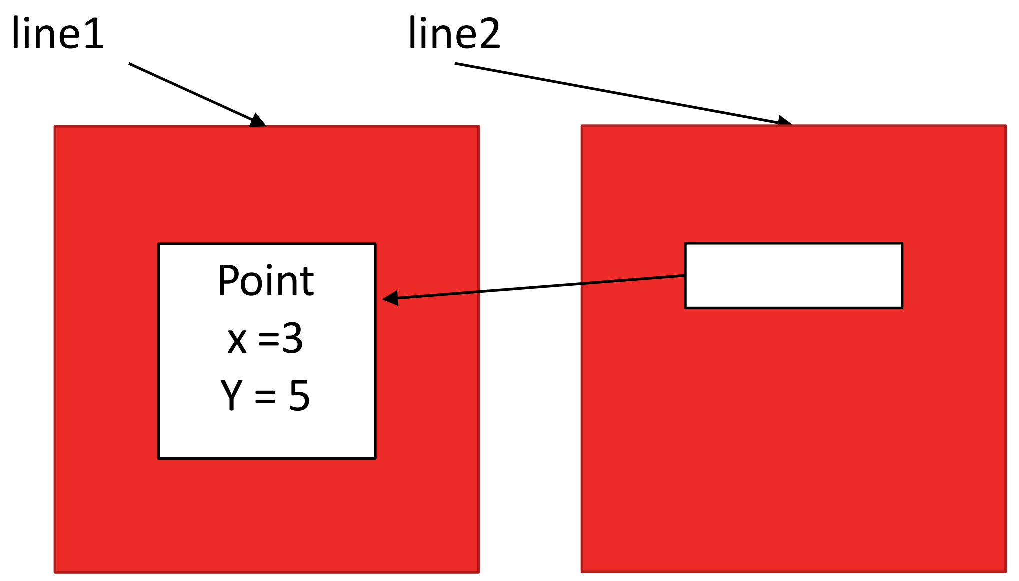
- Additional issues arise when your instance variables are objects and not primitive types.
- What should happen when you want to copy an object containing another object?



line1 = line2

Data hiding and objects

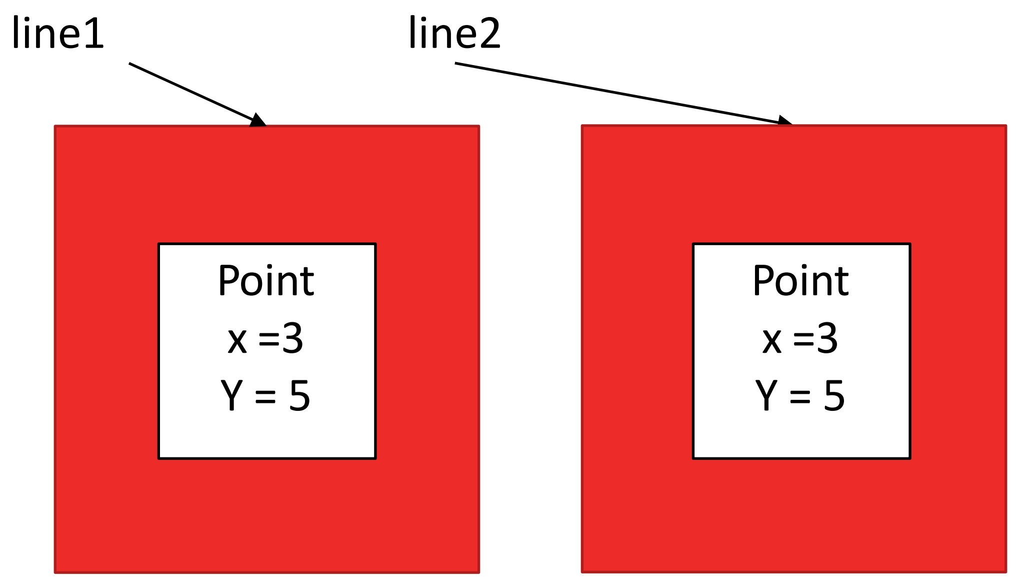
- Additional issues arise when your instance variables are objects and not primitive types.
- What should happen when you want to copy an object containing another object?



line1 = line2.shallowcopy()

Data hiding and objects

- Additional issues arise when your instance variables are objects and not primitive types.
- What should happen when you want to copy an object containing another object?



line1 = line2.deepcopy()

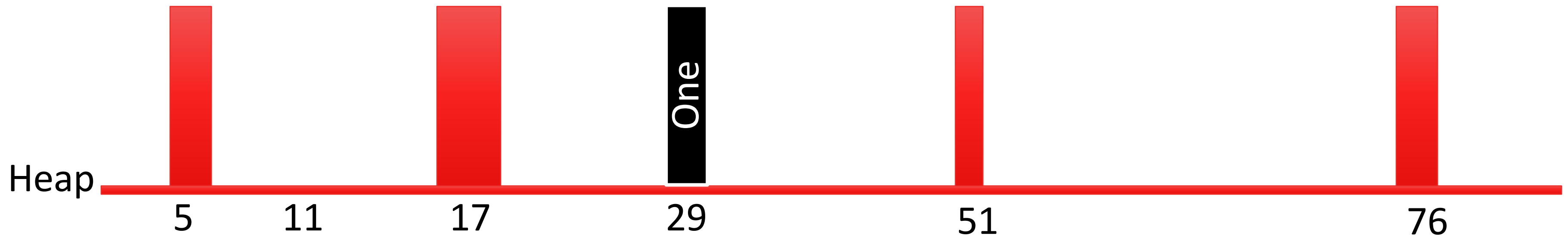
Basic Heap

An example

```
public class Book {
        private String name;
        public Book(String aName){…}
        public String getName();
}
```

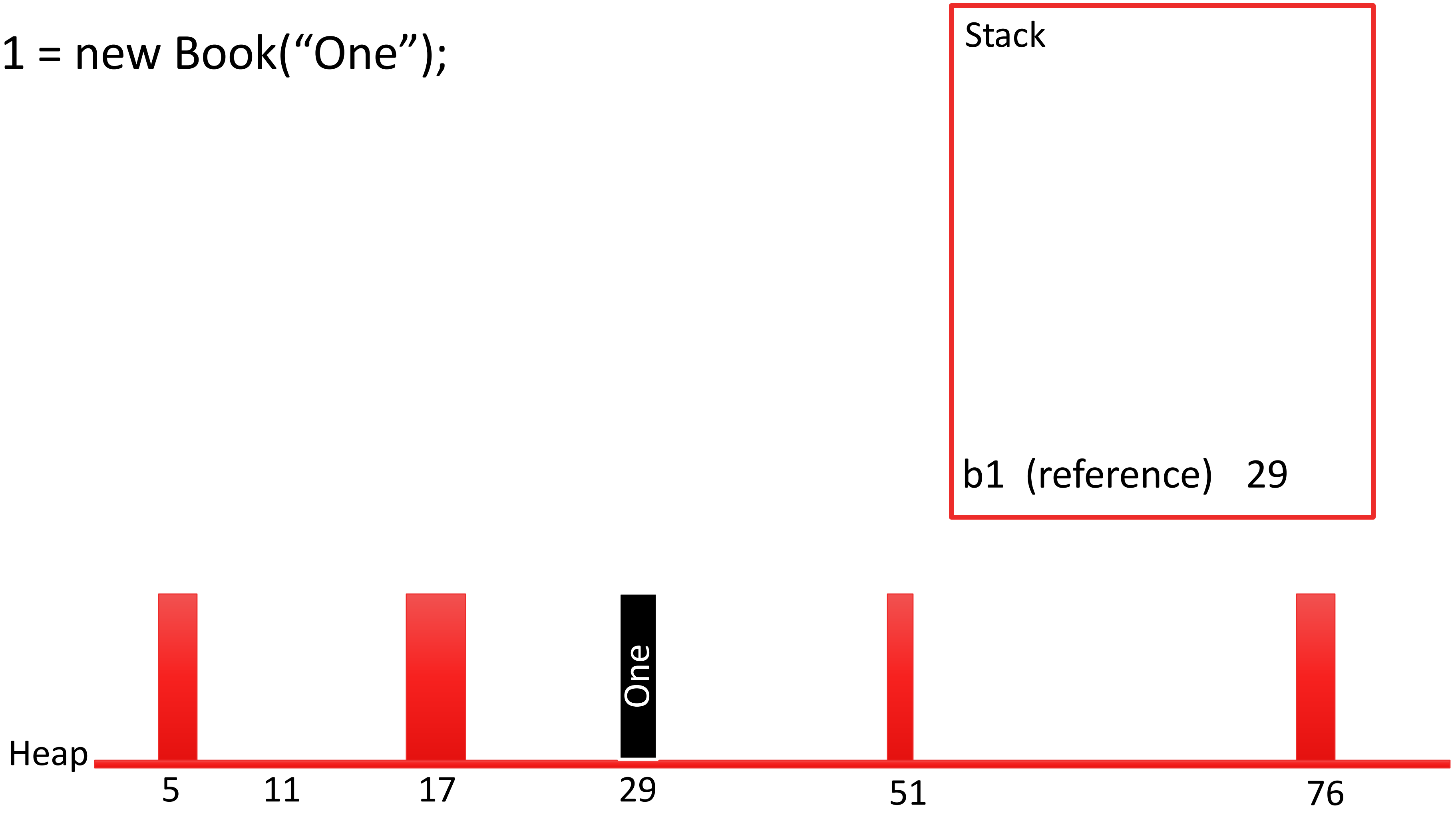
Stack, Heap and References

- Book b1 = new Book(“One”);



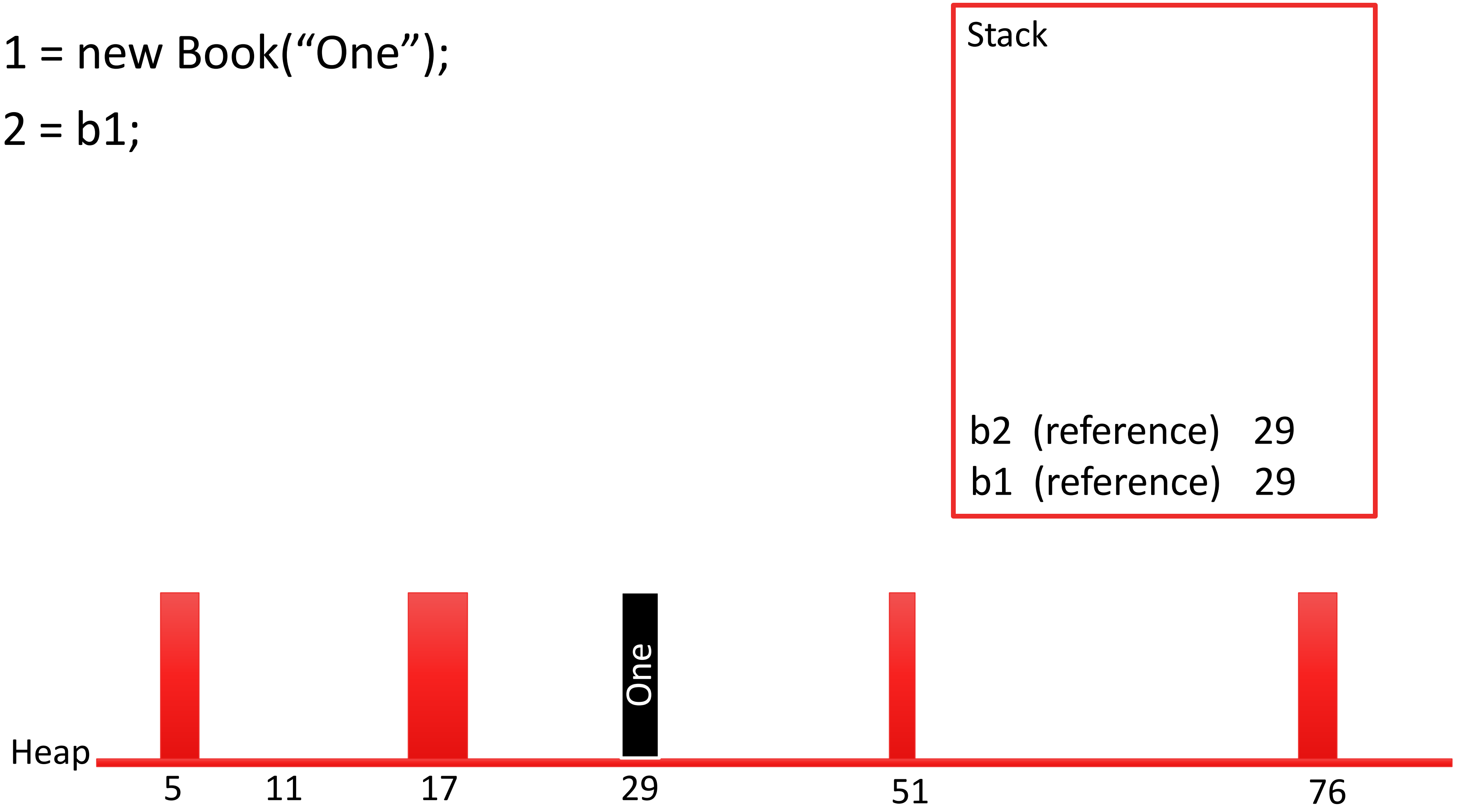
Stack, Heap and References

- Book b1 = new Book(“One”);



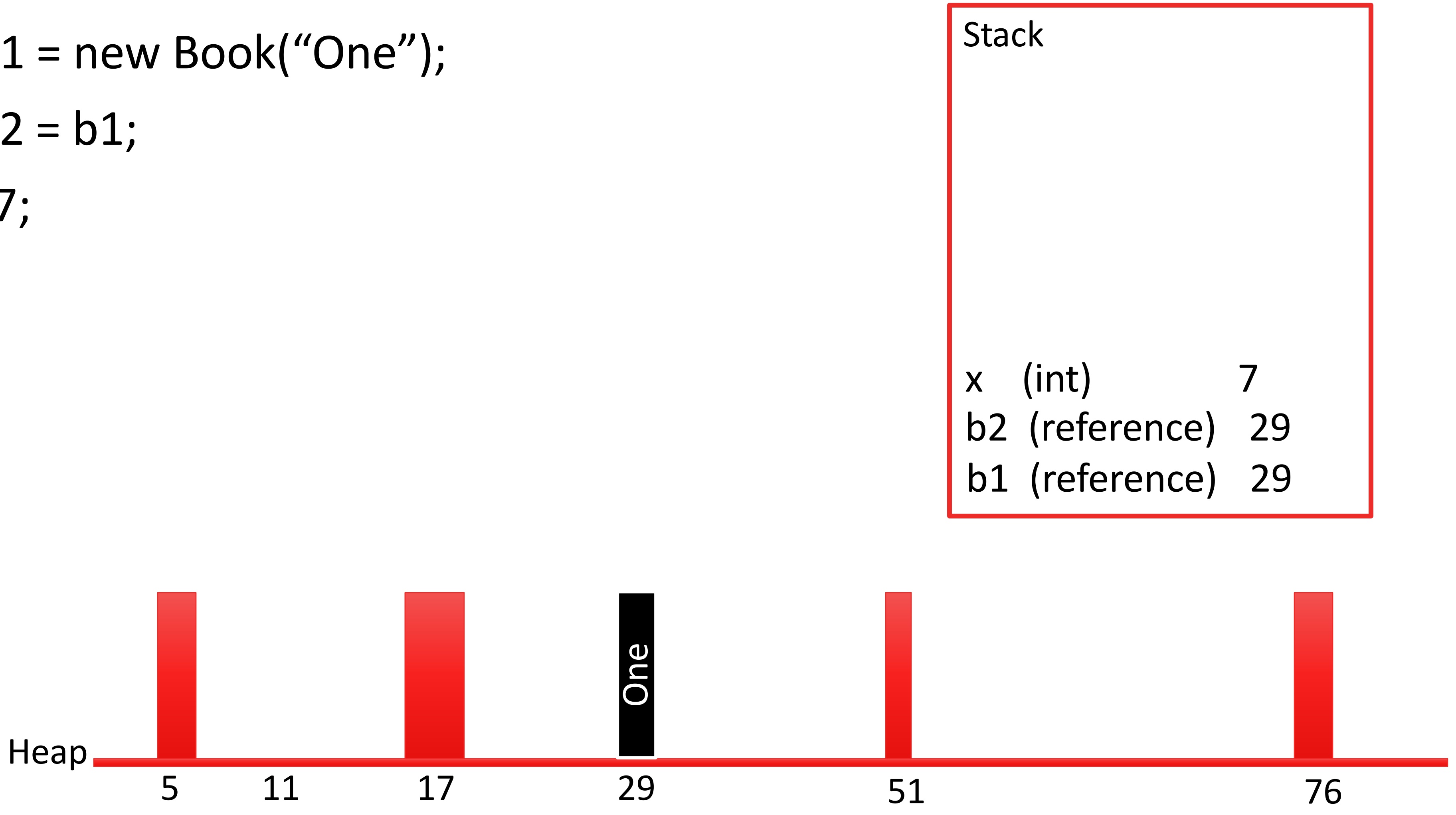
Stack, Heap and References

- Book b1 = new Book(“One”);
- Book b2 = b1;



Stack, Heap and References

- Book b1 = new Book(“One”);
- Book b2 = b1;
- int x = 7;

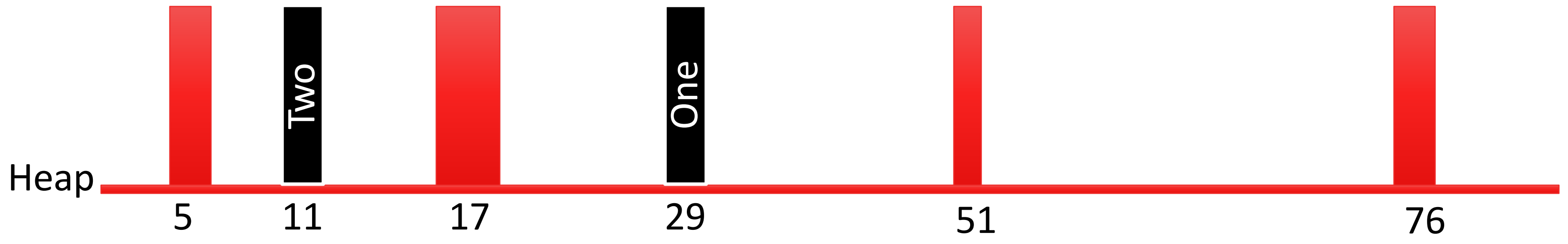


Stack, Heap and References

- Book b1 = new Book(“One”);
- Book b2 = b1;
- int x = 7;
- b1 = new Book(“Two”)

Stack

x	(int)	7
b2	(reference)	29
b1	(reference)	~~29~~ 11

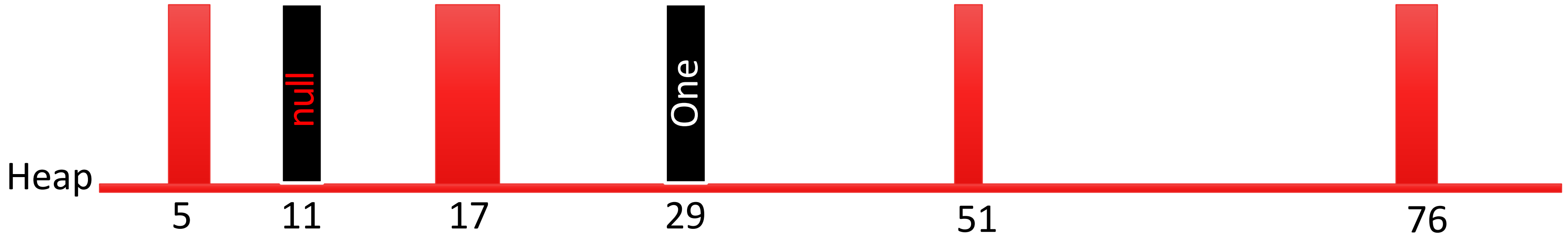


Violation of Encapsulation

- Book b1 = new Book(“One”);
- Book b2 = b1;
- int x = 7;
- b1 = new Book(“Two”)
- b1.name = null;   //if field name was public

Stack

x	(int)	7
b2	(reference)	29
b1	(reference)	~~29~~ 11



Data/Privacy Leak

(others can change my internal data out of my control)

Instance Variables as References

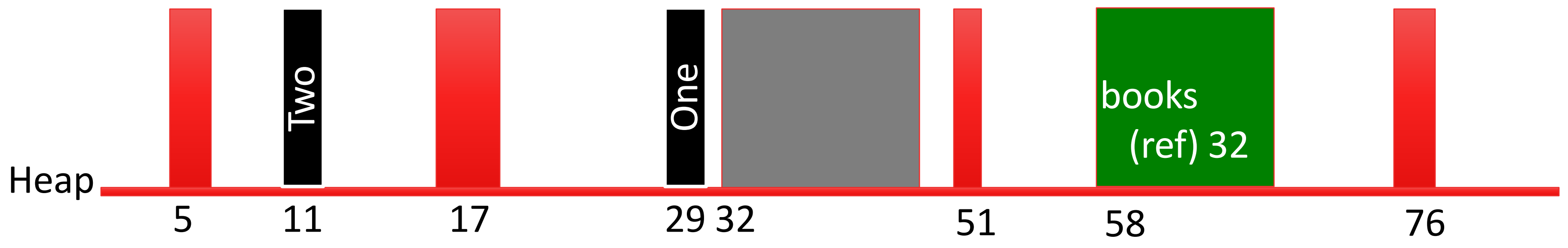
```
public class Book {
        private String name;
        public Book(String aName){…}
        public String getName();
        public void setName( String aName){…}
}

public class Series {
        private ArrayList<Book> books = new ArrayList<Book>();
        public void add(Book aBook){books.add(aBook);}
        public void get(int number) { return books.get(number);}
        public ArrayList<Book> getAll() {return books;}
}
```

Stack, Heap and References

- Series s = new Series();

Stack

s	(reference)	58
x	(int)	7
b2	(reference)	29
b1	(reference)	11

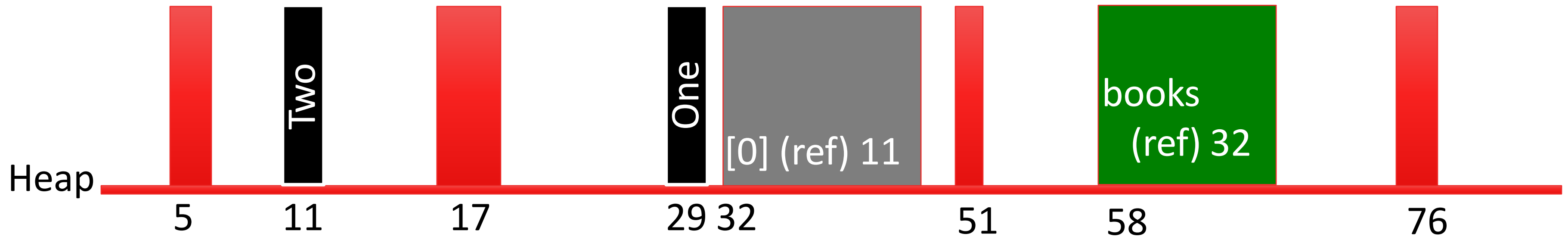


Stack, Heap and References

- Series s = new Series();
- s.add(b1);

Stack

s	(reference)	58
x	(int)	7
b2	(reference)	29
b1	(reference)	11

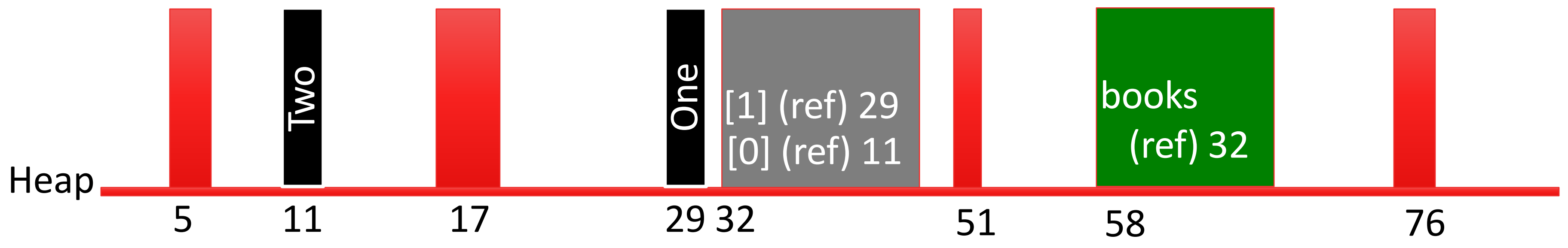


Stack, Heap and References

- Series s = new Series();
- s.add(b1);
- s.add(b2);

Stack

s	(reference)	58
x	(int)	7
b2	(reference)	29
b1	(reference)	11



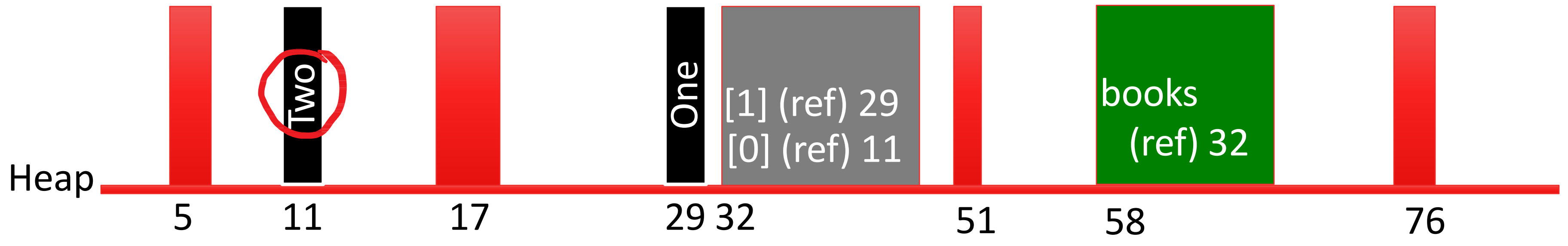
Data Leak 1

- Series s = new Series();
- s.add(b1);
- s.add(b2);
- b1.setName("Boogers"); //I still have access to b1

Stack

al	(reference)	32
s	(reference)	58
x	(int)	7
b2	(reference)	29
b1	(reference)	11

Data Leak!



Copy of Reference versus Copy of Object

- Reference is copied for
    - Assignment
    - Parameter passing
    - Returns from methods

- If the someone else gives our method an object, they still have the reference to it themselves and can modify it as they want, even as I try and use it for my own purposes.

- Available Change -> make our method copy the object instead of using reference.

Removing Privacy Leaks

```
public class Series {
        private ArrayList<Book> books = new ArrayList<Book>();


        public void add(Book aBook){
                books.add(aBook);
    }


        public ArrayList<Book> getAll() {
                return books;
        }
}
```

Reference to a book – whoever calls this method has a reference to the same object.

Removing Privacy Leaks

```
public class Series {
        private ArrayList<Book> books = new ArrayList<Book>();


        public void add(Book aBook){
                books.add(new Book(aBook));
    }


        public ArrayList<Book> getAll() {
                return books;
        }
}
```

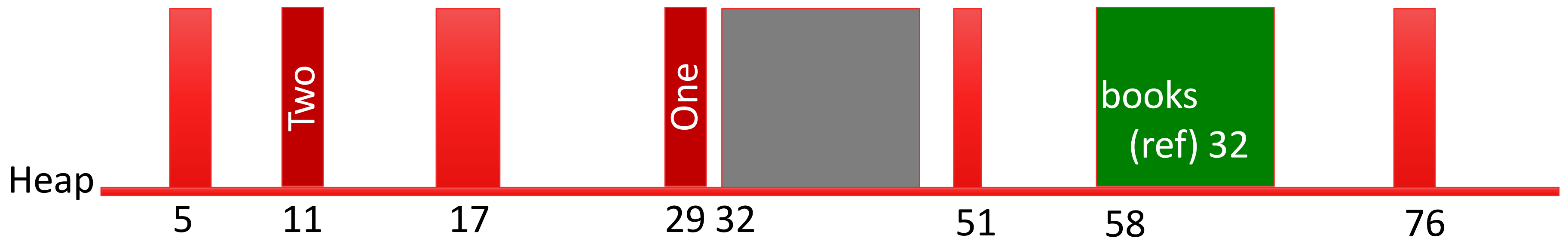
Create a new book and don't share the reference

```
public class Series {
        private ArrayList<Book> books = new ArrayList<Book>();
        public void add(Book aBook){ books.add(new Book(aBook)); }
        public void get(int number) {
                return books.get(number);
        }
        public ArrayList<Book> getAll() {return books;}
}
```

- Series s = new Series();
- s.add(b1);

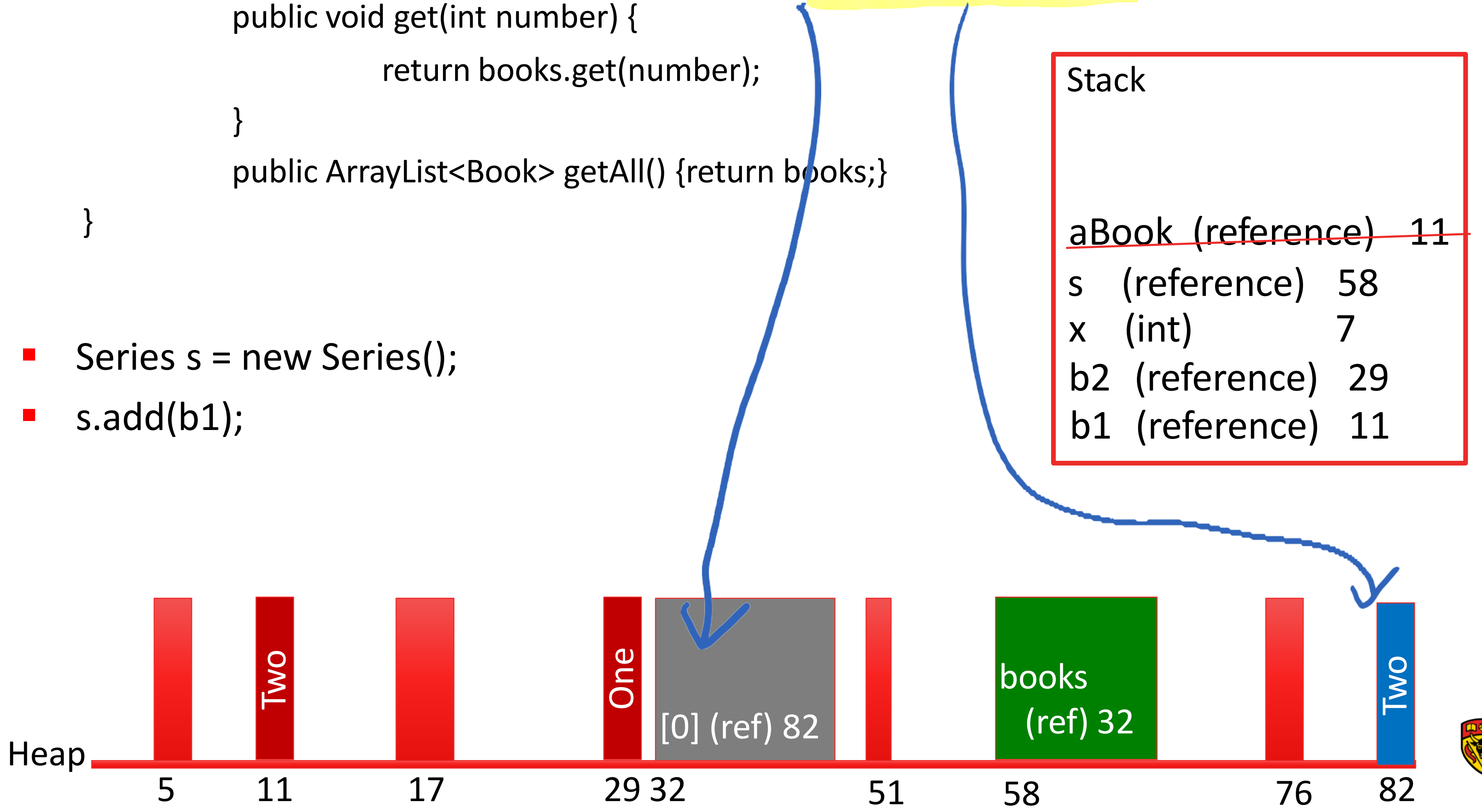
Stack

aBook	(reference)	11
s	(reference)	58
x	(int)	7
b2	(reference)	29
b1	(reference)	11



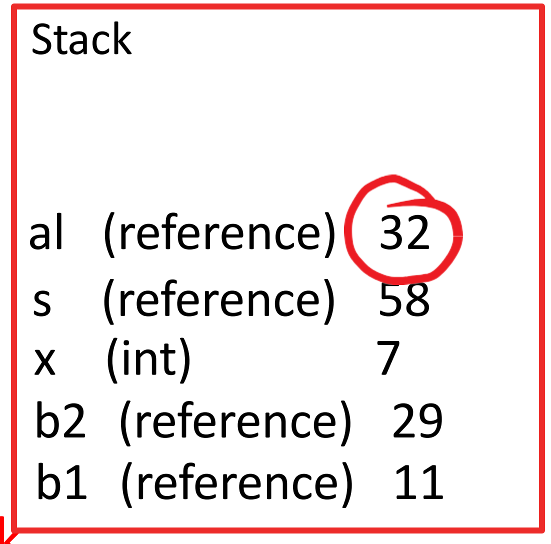
```
public class Series {
        private ArrayList<Book> books = new ArrayList<Book>();
        public void add(Book aBook){ books.add(new Book(aBook)); }
        public void get(int number) {
                return books.get(number);
        }
        public ArrayList<Book> getAll() {return books;}
}
```

- Series s = new Series();
- s.add(b1);



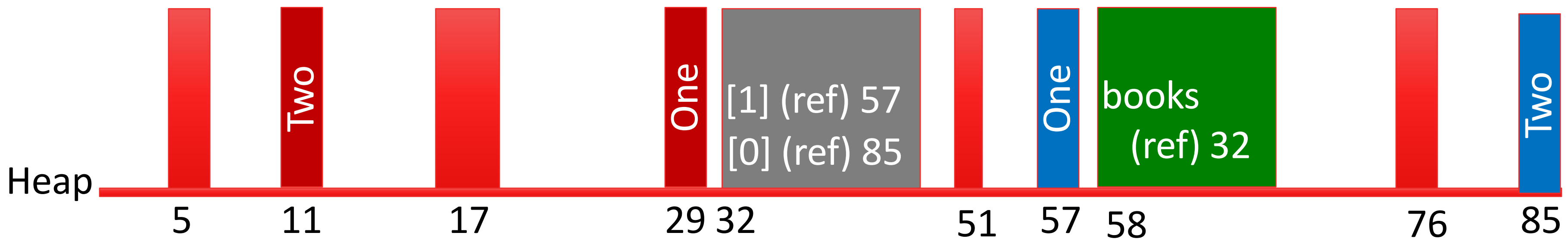
```
public class Series {
        private ArrayList<Book> books = new ArrayList<Book>();
        public void add(Book aBook){ books.add(new Book(aBook)); }
        public void get(int number) {
                return books.get(number);
        }
        public ArrayList<Book> getAll() {return books;}
}
```

- Series s = new Series();
- s.add(b1);
- ArrayList al = s.getAll(); // I could delete any book



Stack

al	(reference)	32
s	(reference)	58
x	(int)	7
b2	(reference)	29
b1	(reference)	11



Onward to … Inheritance

Jonathan Hudson
jwhudson@ucalgary.ca
https://pages.cpsc.ucalgary.ca/~jwhudson/

UNIVERSITY OF CALGARY