Objects and State

COMP1400 Week 9

Mutator methods

The internal state of an object can change.

We do this by changing the values contained in its fields.

Methods that change an object's state are called mutator methods.

Vending machine

Consider a vending machine that issues books.

There is a fixed price for buying a book.

You add credit to the machine by inserting coins.

When you have enough credit, you can buy a book.

State

The state consists of:

- The available stock in the machine (a list of books)
- The current credit balance
- The total amount of money collected.

Accessor methods

We will need accessor methods to:

- get the current stock
- get the current credit balance
- get the total money collected

Mutator Methods

The mutator methods we will implement:

- Add credit
- Buy a book
- Refund (remaining) credit
- Add a book to the stock

Fields

```
private ArrayList<Book> myBooks;

private int myCredit;

private int myTotalCash;

public static final int
                    BOOK_PRICE = 100;
```

Constructor

```
public VendingMachine() {
    // initially empty
    myBooks =
      new ArrayList<Book>();
    myCredit = 0;
    myTotalCash = 0;
}
```

Accessor methods

```
public ArrayList<Book> getBooks();

public int getCredit();

public int getTotalCash();
```

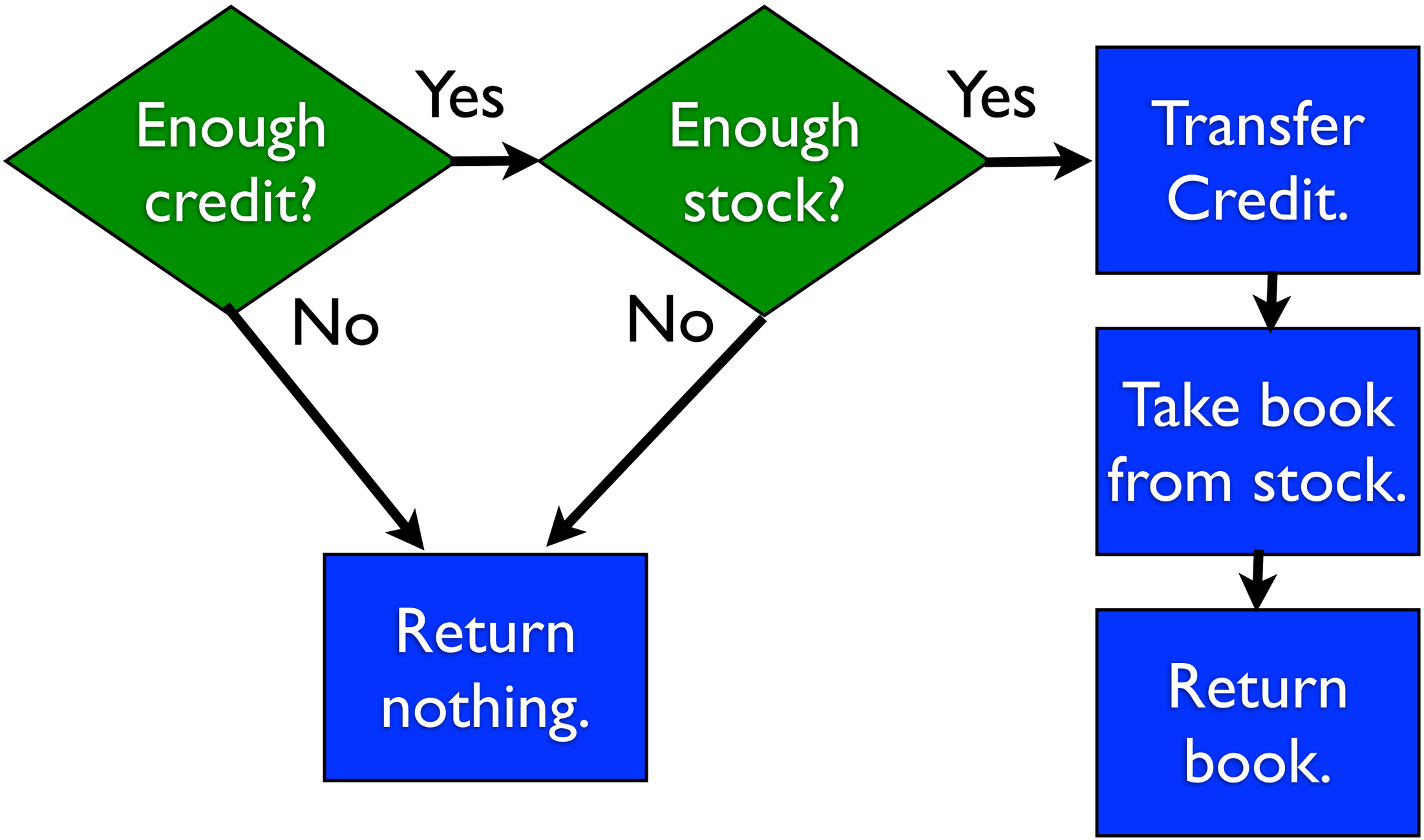
Mutator methods

```
public void addCredit(int amount);

public Book buyBook();

public int refundCredit();

public void addBook();
```

addCredit

```
public void addCredit(int amount)
{
    // ignore attempts to add
    // negative credit

    if (amount > 0) {
        myCredit += amount;
    }
}
```

buyBook flow chart



buyBook

```
public Book buyBook() {
    // 1. check for enough credit
    if (myCredit < BOOK_PRICE) {
        return null;
    }
    // cont...
```

buyBook

```
// 2. check there is stock
if (myBooks.isEmpty()) {
    return null;
}
// cont...
```

buyBook

```
// 3. transfer book price
myCredit -= BOOK_PRICE;
myTotalCash += BOOK_PRICE;
// cont...
```

buyBook

```
        // 4. remove the first book
        //     in stock

        Book book = myBooks.remove(0);

        // 5. return it

        return book;

    }
```

The null object

The keyword `null` is used to represent the null object.

The null object is the way Java represents "nothing".

The null object

The null object can belong to any object type but not a primitive type:

```
String s = null;

Book b = null;

Integer i = null;

int i2 = null;  // WRONG!
```

The null object

Note the difference between the empty string and the null string:

```
String empty = "";
String nothing = null;
// empty != nothing
```

The null object

It is an error to try to call a method on the null object:

```
Book b = null;
String t = b.getTitle();
// NullPointerException
```

addBook

```
public void addBook(Book book) {
    // don't add the null book
    if (book != null) {
        myBooks.add(book);
    }
}
```

Breaking encapsulation

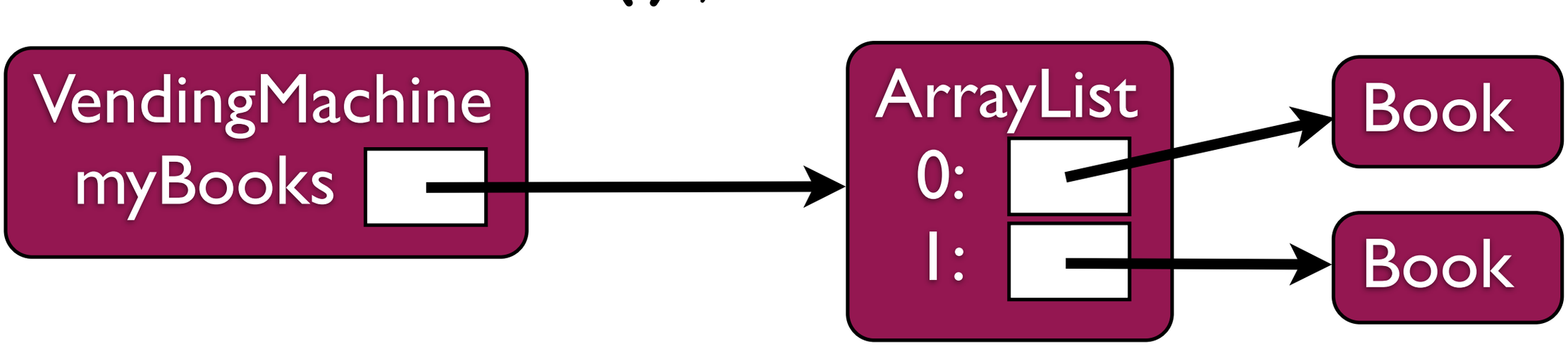
Simply making a field private isn't always good enough to maintain encapsulation.

Consider:

```
public ArrayList<Book> getBooks()
{
    return myBooks;
}
```

Breaking encapsulation

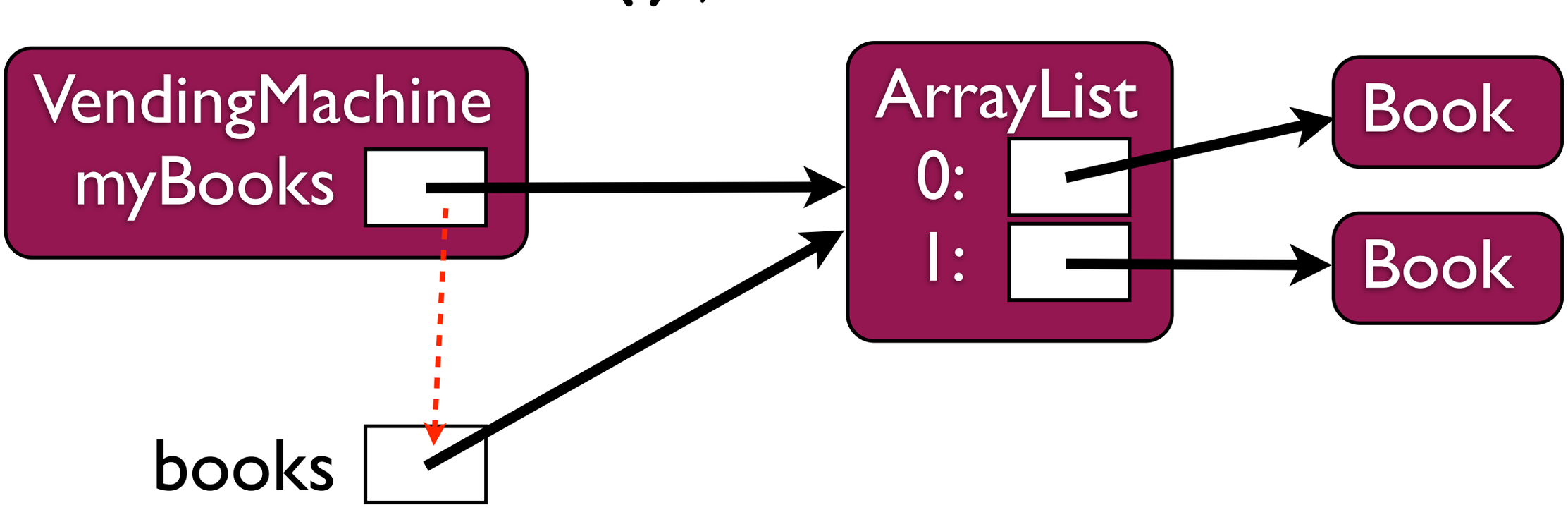
```
ArrayList<Book> books =
vendingMachine.getBooks();

books.clear();
```



books

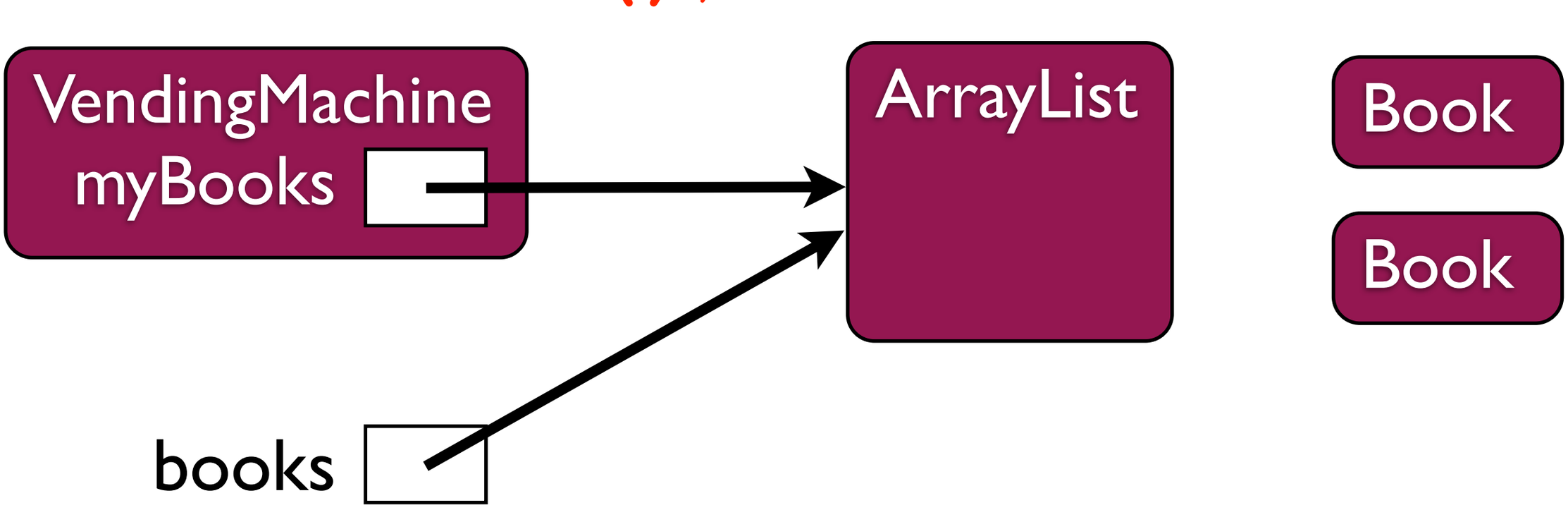
Breaking encapsulation

```
ArrayList<Book> books =
vendingMachine.getBooks();

books.clear();
```



Breaking encapsulation

```
ArrayList<Book> books =
vendingMachine.getBooks();

books.clear();
```



Two solutions

There are two simple solutions to this problem:

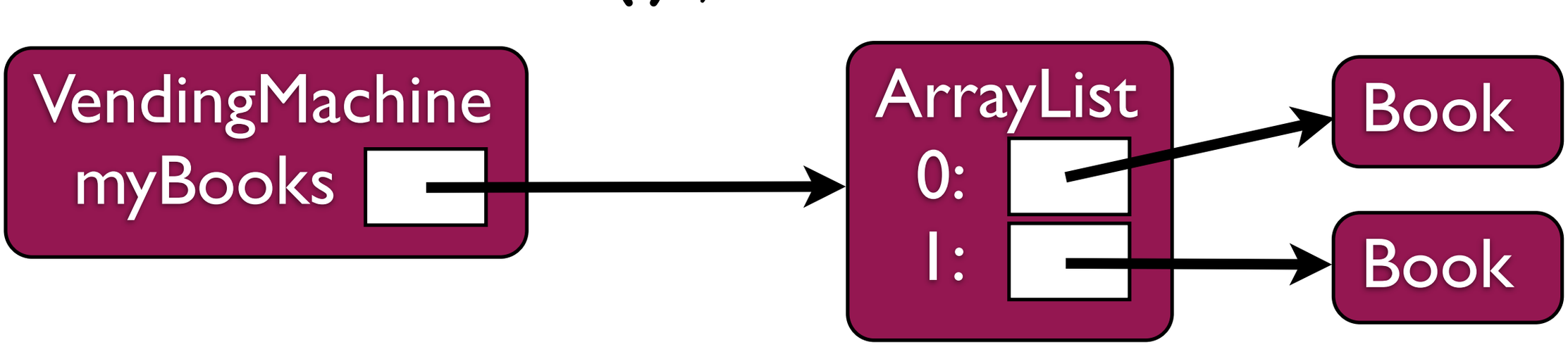
1. return a copy of the list, not a reference
2. wrap the list in a layer that prevents modification

Solution I

```
public ArrayList<Book> getBooks()
{
    // return a copy of myBooks

    return new
        ArrayList<Book>(myBooks);

}
```

Solution 1

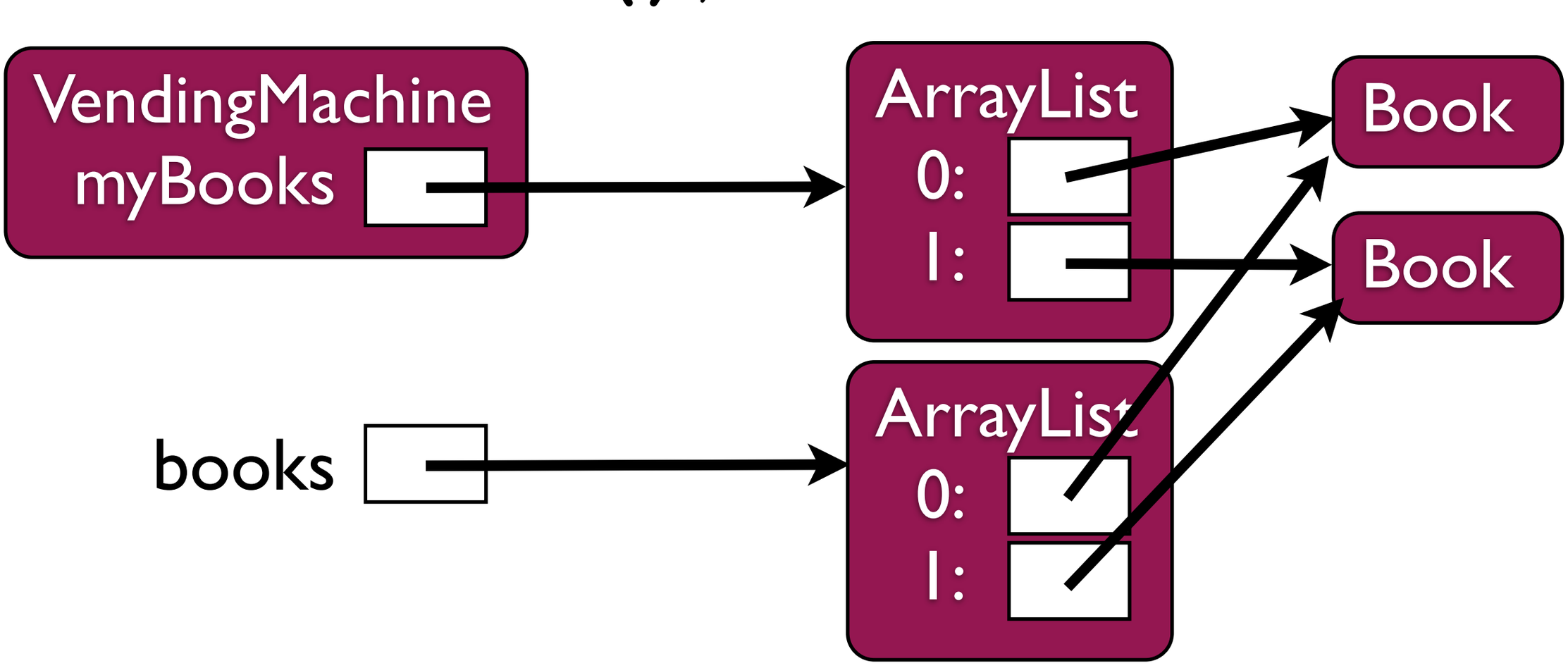
```
ArrayList<Book> books =
vendingMachine.getBooks();

books.clear();
```



books

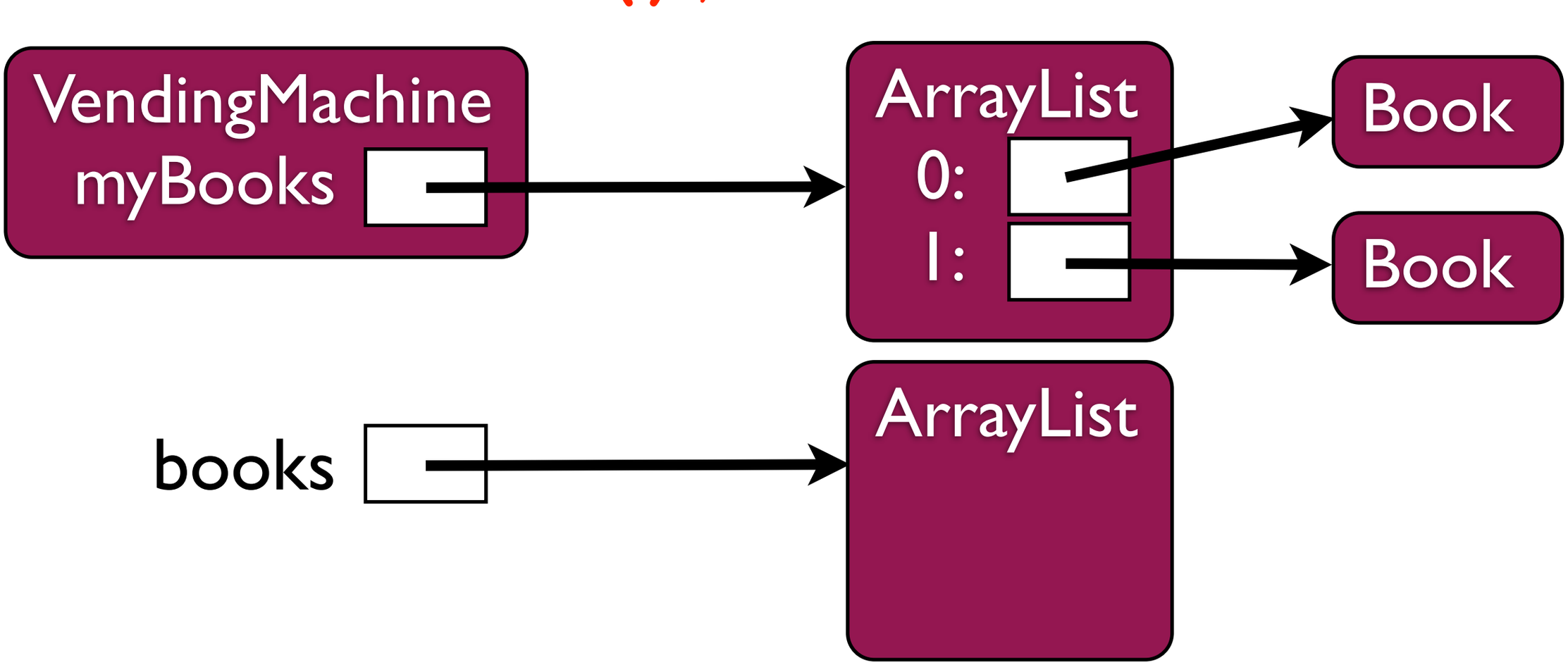
Solution 1

```
ArrayList<Book> books =
vendingMachine.getBooks();

books.clear();
```



Solution I

```
ArrayList<Book> books =
vendingMachine.getBooks();

books.clear();
```



Solution 1

Advantages:

- returns a list that can be used without affecting the vendingMachine

Disadvantages:

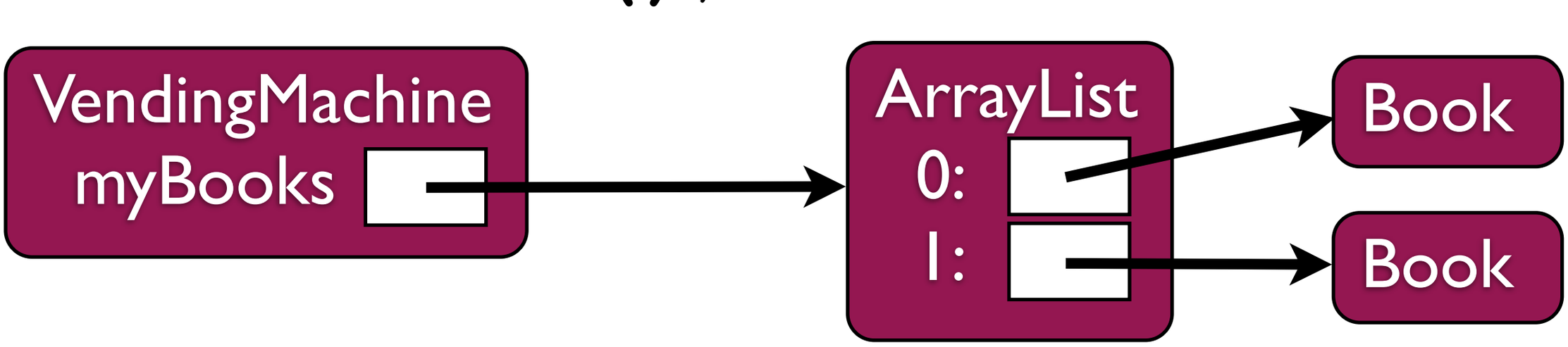
- copying on every access is time consuming

Solution 2

```
public ArrayList<Book> getBooks()
{
    // return an unmodifiable
    // version of myBooks

    return
     Collections.unmodifiableList(
                              myBooks);
}
```

Solution 2

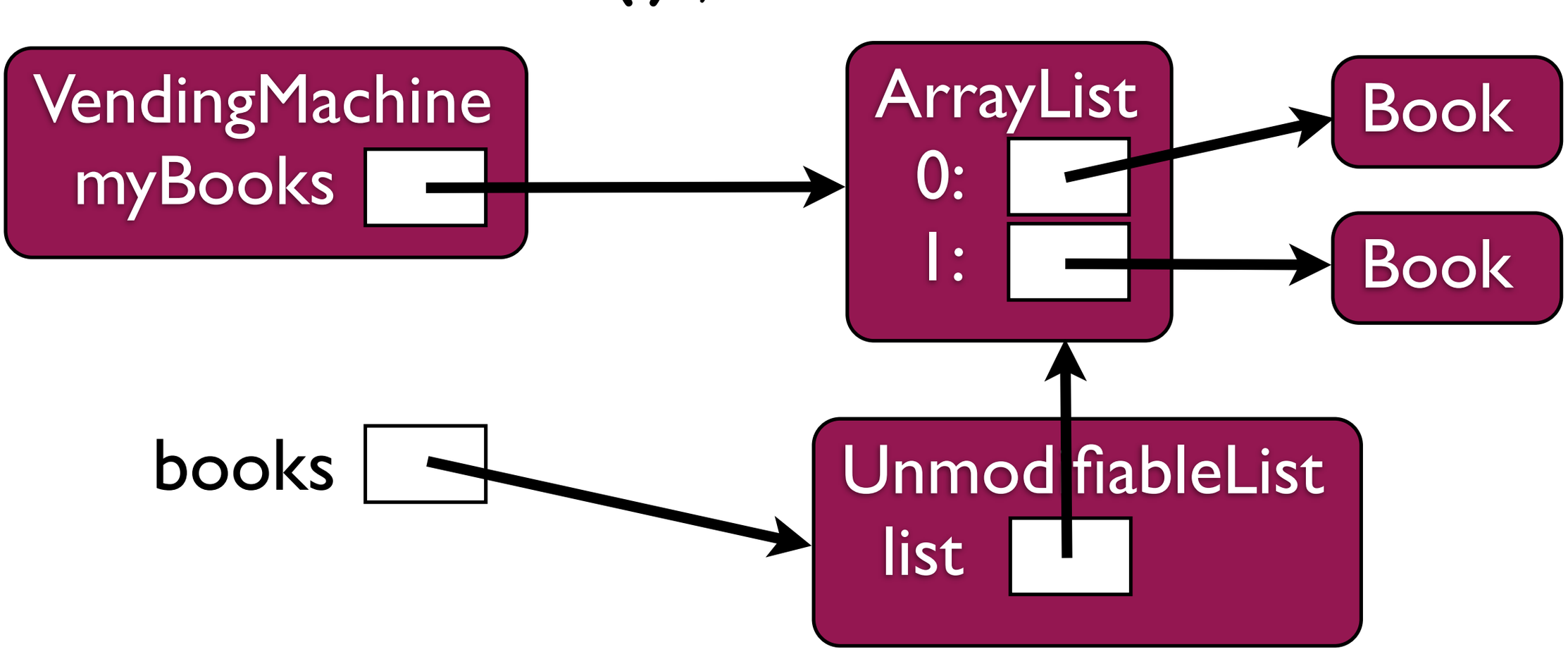
```
ArrayList<Book> books =
vendingMachine.getBooks();

books.clear();
```



books

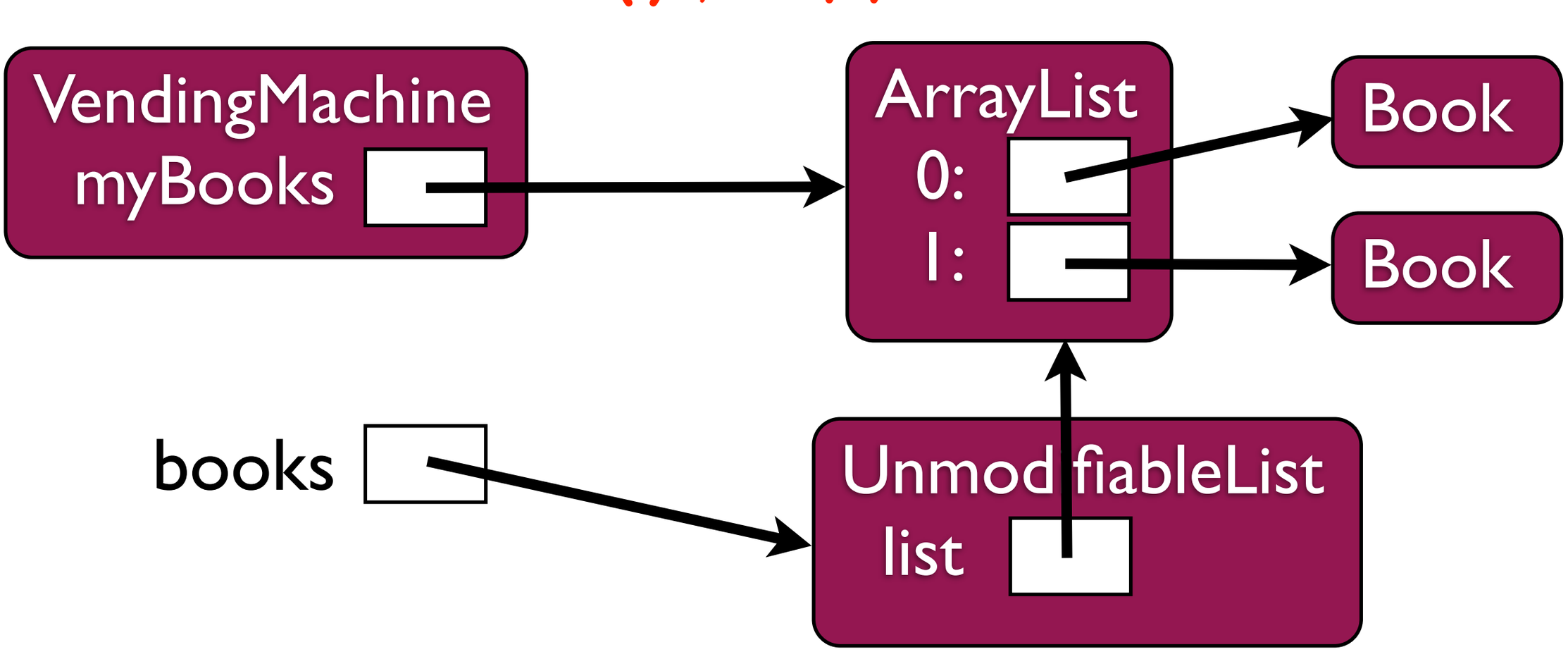
Solution 2

```
ArrayList<Book> books =
vendingMachine.getBooks();

books.clear();
```



Solution 2

```
ArrayList<Book> books =
vendingMachine.getBooks();

books.clear();   // ERROR
```



Solution 2

Advantages:

- prevents outsiders modifying myBooks
- avoids cost of copying

Disadvantages:

- users need to copy the list themselves in order to modify it