CS 61B Spring 2022 — Objects

Exam Prep Discussion 4: February 7, 2022

1 Give em the 'Ol Switcheroo

For each function call in the `main` method, write out the x and y values of both foobar and baz after executing that line. (Spring '15, MT1)

```
1  public class Foo {
2      public int x, y;
3
4      public Foo (int x, int y) {
5          this.x = x;
6          this.y = y;
7      }
8      public static void switcheroo (Foo a, Foo b) {
9          Foo temp = a;
10         a = b;
11         b = temp;
12     }
13     public static void fliperoo (Foo a, Foo b) {
14         Foo temp = new Foo(a.x, a.y);
15         a.x = b.x;
16         a.y = b.y;
17         b.x = temp.x;
18         b.y = temp.y;
19     }
20     public static void swaperoo (Foo a, Foo b) {
21         Foo temp = a;
22         a.x = b.x;
23         a.y = b.y;
24         b.x = temp.x;
25         b.y = temp.y;
26     }
27
28     public static void main (String[] args) {
29         Foo foobar = new Foo(10, 20);
30         Foo baz = new Foo(30, 40);
31         switcheroo(foobar, baz);    foobar.x: ___ foobar.y: ___ baz.x: ___ baz.y: ___
32         fliperoo(foobar, baz);      foobar.x: ___ foobar.y: ___ baz.x: ___ baz.y: ___
33         swaperoo(foobar, baz);      foobar.x: ___ foobar.y: ___ baz.x: ___ baz.y: ___
34     }
35 }
```

Solution:

```
line 34: foobar.x: 10 foobar.y: 20 baz.x: 30 baz.y: 40
line 35: foobar.x: 30 foobar.y: 40 baz.x: 10 baz.y: 20
line 36: foobar.x: 10 foobar.y: 20 baz.x: 10 baz.y: 20
```

Here is a video walkthrough of the solutions for this problem.

Explanation:

switcheroo: Note that `switcheroo` assigns a local variable `temp` to a, but never mutates objects, e.g. by reassigning `a.x`. This means that all `switcheroo` does is move around its local pointers to `temp`, `a`, and `b`; nothing in `foobar` or `baz` is actually changed.

fliperoo: Here, `a` points to `foobar` and `b` points to `baz`. `temp` refers to an object with the same initial `x` and `y` values as `a`, which are `10` and `20` respectively. Lines 15 and 16 change `foobar` to have {`x: 30, y: 40`}. Then, lines 17 and 18 allow `baz` to take on the same `x` and `y` values as `temp`, which are {`x: 10, y: 20`}.

swaperoo: In `swaperoo`, instead of creating a new object, we simply point `temp` to the same object as `a`. In lines 22 and 23, we override `foobar`'s `x` and `y` values to become the same as `baz`'s: {`x: 10, y: 20`}. In line 24 and 25, we assign `baz`'s `x` and `y` values to be equal to `temp`'s. But remember, `temp` is pointing to the same object as `a`, which points to `foobar`, and which we just modified to have {`x: 10, y: 20`}. Thus, `baz` does not change.

2 IntList to Array

For this problem we will implement a version of `arraycopy` that copies elements from an IntList into an array of `int`s. As a reminder, here is the `arraycopy` method:

```
System.arraycopy(Object src, int sourcePos, Object dest, int destPos, int len)
```

`System.arraycopy` copies `len` elements from array `src` (starting at index `source`) to array `destArr` (starting from index `dest`).

To simplify things, let's restrict ourselves to using only `int[]`, and assume that `srcList` and `destArr` are not null. Additionally, assume that `sourcePos`, `destPos`, and `len` will not cause an IndexOutOfBoundsException to be thrown.

For example, let IntList `L` be `(1 -> 2 -> 3 -> 4 -> 5)` and `int[] arr` be an empty array of length 3. Calling `arrayCopyFromIntList(L, 1, arr, 0, 3)` will result in `arr={2, 3, 4}`.

```
/** Works just like System.arraycopy, except srcList is of type IntList. */
public static void arrayCopyFromIntList(IntList srcList, int sourcePos,
        int[] destArr, int destPos, int len) {

    for ( ___________; ___________; ___________ ) {

        ________________________ = ____________________________________;
    }

    for ( ___________; ___________; ___________ ) {

        ________________________ = ____________________________________;

        ________________________ = ____________________________________;

    }
}
```

Solution:

```
/** Works just like System.arraycopy, except srcList is of type IntList. */
public static void arrayCopyFromIntList(IntList srcList, int sourcePos,
        int[] destArr, int destPos, int len) {
    for (int i = 0; i < sourcePos; i += 1) {
        srcList = srcList.tail;
    }

    for (int i = destPos; i < destPos + len; i += 1) {
        destArr[i] = srcList.head;
        srcList = srcList.tail;
    }
}
```

Here is a video walkthrough of the solutions for this problem. **Explanation:** `arrayCopyFromIntList` should copy over `len` items from our source `IntList` to our destination array, starting at `sourcePos` in the source `IntList` and `destPos` in the destination array.
In the first loop, we move along the `srcList` to get the correct starting position. In the second loop, we copy over `len` items from the `srcList`, starting at `destpos` in the array.

3 Static Books

Suppose we have the following `Book` and `Library` classes.

```
class Book {
    public String title;
    public Library library;
    public static Book last = null;

    public Book(String name) {
        title = name;
        last = this;
        library = null;
    }

    public static String lastBookTitle() {
        return last.title;
    }
    public String getTitle() {
        return title;
    }
}
```

```
class Library {
    public Book[] books;
    public int index;
    public static int totalBooks = 0;

    public Library(int size) {
        books = new Book[size];
        index = 0;
    }

    public void addBook(Book book) {
        books[index] = book;
        index++;
        totalBooks++;
        book.library = this;
    }
}
```

(a) For each modification below, determine whether the code of the `Library` and `Book` classes will compile or error if we **only** made that modification, i.e. treat each modification independently.

1. Change the `totalBooks` variable to **non static**
2. Change the `lastBookTitle` method to **non static**
3. Change the `addBook` method to **static**
4. Change the `last` variable to **non static**
5. Change the `library` variable to **static**

Solution:

Here is a video walkthrough of the solutions for this part and the next.

1. Compile
   `totalBooks` is only used inside of a nonstatic function, so changing it to nonstatic would not cause compilation errors (although note that it no longer counts the total number of books correctly).
2. Compile
   Both static and nonstatic methods can access static variables, so changing `lastBookTitle` to be static would still allow it to access `last.title`.
3. Error
   Static methods cannot access instance variables, so changing `addBook` to be static would cause it to be unable to find the `books` or `index` variables.

4. Error
   Again, static methods cannot access instance variables, so changing `last` to be static would cause `lastBookTitle` to fail.

5. Compile
   Constructors are allowed to modify static variables; similarly, instances of a class can access that class's static variables. Thus, changing `library` to be static would not affect the `Book` constructor or `book.library` in `addBook`.

(b) Using the `Book` and `Library` classes from before, write the output of the `main` method below. If a line errors, put the precise reason it errors and continue execution.

```
1  public class Main {
2      public static void main(String[] args) {
3          System.out.println(Library.totalBooks);                  _____________________
4          System.out.println(Book.lastBookTitle());                _____________________
5          System.out.println(Book.getTitle());                     _____________________
6
7          Book goneGirl = new Book("Gone Girl");
8          Book fightClub = new Book("Fight Club");
9
10         System.out.println(goneGirl.title);                      _____________________
11         System.out.println(Book.lastBookTitle());                _____________________
12         System.out.println(fightClub.lastBookTitle());           _____________________
13         System.out.println(goneGirl.last.title);                 _____________________
14
15         Library libraryA = new Library(1);
16         Library libraryB = new Library(2);
17         libraryA.addBook(goneGirl);
18
19         System.out.println(libraryA.index);                      _____________________
20         System.out.println(libraryA.totalBooks);                 _____________________
21
22         libraryA.totalBooks = 0;
23         libraryB.addBook(fightClub);
24         libraryB.addBook(goneGirl);
25
26         System.out.println(libraryB.index);                      _____________________
27         System.out.println(Library.totalBooks);                  _____________________
28         System.out.println(goneGirl.library.books[0].title);     _____________________
29     }
30 }
```

Solution:

```
1  public class Main {
2      public static void main(String[] args) {
3          System.out.println(Library.totalBooks);                  0
4          System.out.println(Book.lastBookTitle());                Error, NullPointerException
5          System.out.println(Book.getTitle());                     Error, does not compile
6
7          Book goneGirl = new Book("Gone Girl");
8          Book fightClub = new Book("Fight Club");
9
10         System.out.println(goneGirl.title);                      Gone Girl
11         System.out.println(Book.lastBookTitle());                Fight Club
```

```
12          System.out.println(fightClub.lastBookTitle());          Fight Club
13          System.out.println(goneGirl.last.title);                Fight Club
14
15          Library libraryA = new Library(1);
16          Library libraryB = new Library(2);
17          libraryA.addBook(goneGirl);
18
19          System.out.println(libraryA.index);                     1
20          System.out.println(libraryA.totalBooks);                1
21
22          libraryA.totalBooks = 0;
23          libraryB.addBook(fightClub);
24          libraryB.addBook(goneGirl);
25
26          System.out.println(libraryB.index);                     2
27          System.out.println(Library.totalBooks);                 2
28          System.out.println(goneGirl.library.books[0].title);    Fight Club
29      }
30  }
```

Explanation:

Line 3: The static variable `totalBooks` is initialized to 0.

Line 4: We haven't created any books yet, so the `Book` constructor has never been called, and `last` is `null`. When we attempt to call `lastBookTitle`, we access the `title` property of a `null` object, which results in a `NullPointerException`.

Line 5: You cannot call a nonstatic method using the class name; only instances of the class can call their instance methods.

Line 10: The string `"Gone Girl"` was passed into the constructor of the `goneGirl` object, so its title is `Gone Girl` (printing removes quotes).

Line 11: Whenever a new book is created, the static variable `last` points to it. Thus, `last` points to the most recently created book, `fightClub`.

Line 12: Instances of a class can access static variables.
`goneGirl.last` is the same as `Book.last`, which is `fightClub`.

Line 19: `index` gets incremented each time we call `addBook`, so after adding `goneGirl` to `libraryA`, its index is 1.

Line 20: `totalBooks` gets incremented each time we call `addBook`, so after adding `goneGirl` to `libraryA`, its `totalBooks` is 1. (Remember, instances can access a class's static variables).

Line 26: `index` gets incremented each time we call `addBook`, and it is an instance variable, so each library has its own copy of `index`. After adding `goneGirl` and `fightClub` to `libraryB`, its index is 2.

Line 27: `totalBooks` is a static variable, so on line 22, `totalBooks` gets reset to 0 for the entire class. Then, it gets incremented twice in `addBook` for a total of 2.

Line 28: In `addBook`, we set `book.library` equal to the library to which that book was *most recently added to*. `goneGirl` was most recently added

to `libraryB`, so its `library` is `libraryB`. Each library has its own `books` array which tracks books from oldest to newest addition. The first book added to `libraryB` was `fightClub`.