CS 61B Fall 2025

Scope, Pass-By-Value, Static

Exam-Level 01: September 1, 2025

1 Quik Maths

(a) Fill in the blanks in the main method below. (Fall '16, MT1)

```
public class QuikMaths {
    public static void multiplyBy3(int[] A) {
        for (int i = 0; i < A.length; i += 1) {
            int x = A[i];
            x = x * 3;
        }
    }

    public static void multiplyBy2(int[] A) {
        int[] B = A;
        for (int i = 0; i < B.length; i+= 1) {
            B[i] *= 2;
        }
    }

    public static void swap(int A, int B) {
        int temp = B;
        B = A;
        A = temp;
    }

    public static void main(String[] args) {
        int[] arr = new int[]{2, 3, 3, 4};

        multiplyBy3(arr); // Value of arr: {2, 3, 3, 4}


        arr = new int[]{2, 3, 3, 4};

        multiplyBy2(arr); // Value of arr: {4, 6, 6, 8}


        int a = 6;
        int b = 7;

        swap(a, b); // Value of a:  6   Value of b:  7

    }
}
```

Click here for visualizer link!

line 23: `/* Value of arr: {2, 3, 3, 4} */`, because we are changing a copy of each element, not the original elements. The enhanced `for` loop also has a similar effect to this.
line 28: `/* Value of arr: {4, 6, 6, 8} */`, because B and A point to the same underlying array.
line 34: `/* Value of a: 6  Value of b: 7 */`, Java is pass by value, so you are only swapping copies of the original integers.

(b) Now take a look at the code below. How could we write **swap** to perform swapping primitive variables in a function? Be sure to use the **IntWrapper** class below.

```
class IntWrapper {
    int x;
    public IntWrapper(int value) {
        x = value;
    }
}

public class SwapPrimitives {
    public static void main(String[] args) {
        IntWrapper first = new IntWrapper(6);

        IntWrapper second = new IntWrapper(7);


        swap(first, second);
    }


    public static void swap(IntWrapper first, IntWrapper second) {


        int temp = first.x;


        first.x = second.x;


        second.x = temp;
    }

}
```

2 PlanetScanner

Fill in the code below. The PlanetScanner class should have a constructor that takes in a array of **Planet** objects and the number of planets to scan. It should have two methods:

- `scannedPlanets` returns the names of the planets that have been scanned.
- `scanMorePlanets` scans more planets.

For example, if we create a `PlanetScanner` for an array of 10 planets, with numToScan equal to 4, then `scannedPlanets()` should return an array of the names of the first 4 planets. If we then call `scanMorePlanets(3)`, then `scannedPlanets()` should return an array of all 7 planets scanned so far.

As with midterm problems, you may not need all available space.

This problem has many possible solutions. This is the cleanest one Dawn could think of.

```
public class Planet {
  // Implementation ommitted...
}

public class PlanetScanner {


    private Planet[] allPlanets;

    private Planet[] scannedPlanets;

    public PlanetScanner(Planets[] planets, int numToScan) {

        this.allPlanets = planets;
        this.scannedPlanets = new int[0];
        this.scanMorePlanets(numToScan);

    }

    public Planet[] scannedPlanets() {

        return this.scannedPlanets;
    }

    public void scanMorePlanets(int numToScan) {

        int newSize = this.scannedPlanets.length + numToScan;
        Planet[] newScan = new Planet[newSize];

        for (int i = 0; (i < newSize) && (i < allPlanets.length); i++) {
            newScan[i] = this.allPlanets[i];
        }

        this.scannedPlanets = newScan;

    }
}
```

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**

   Compile

2. Change the **lastBookTitle** method to **non static**

   Compile

3. Change the **addBook** method to **static**

   Error, cannot access instance variable books in a static method.

4. Change the **last** variable to **non static**

   Error, cannot access instance variable last in a static method.

5. Change the **library** variable to **static**

   Compile

(b) Using the original **Book** and **Library** classes (i.e., without the modifications from part a), write the output of the **main** method below. If a line errors, put the precise reason it errors and continue execution.

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

        System.out.println(Library.totalBooks);                 0

        System.out.println(Book.lastBookTitle());               RE, NullPointerException

        System.out.println(Book.getTitle());                    CE, does not compile

        Book goneGirl = new Book("Gone Girl");
        Book fightClub = new Book("Fight Club");

        System.out.println(goneGirl.title);                     Gone Girl

        System.out.println(Book.lastBookTitle());               Fight Club

        System.out.println(fightClub.lastBookTitle());          Fight Club

        System.out.println(goneGirl.last.title);                Fight Club

        Library libraryA = new Library(1);
        Library libraryB = new Library(2);
        libraryA.addBook(goneGirl);

        System.out.println(libraryA.index);                     1

        System.out.println(libraryA.totalBooks);                1

        libraryA.totalBooks = 0;
        libraryB.addBook(fightClub);
        libraryB.addBook(goneGirl);

        System.out.println(libraryB.index);                     2

        System.out.println(Library.totalBooks);                 2

        System.out.println(goneGirl.library.books[0].title);    Fight Club
    }
}
```

Click here for visualizer link

4 Helpful LLM Use

On HW2, you had a chance to play around with large language models for solving the **`starTriangle`** problem. You've probably also used LLMs in prior programming classes.

What are some ways that you've used LLMs to help your learning? What are some ays that you've used LLMs that actually hindered your learning? Did you find it helpful to see what an LLM came up with for **`starTriangle`**?