

## WES-CS GROUP MEETING #9

### Exercise 1: Book Festival (Different kinds of Fields and Methods)

This exercise will help you to understand the difference between class and instance fields, between class and instance methods, and between public and private methods.

**Part (a)** Look at the `Book` class defined below. Say which fields are *class* fields and which are *instance* fields. Do the same for the methods.

```
public class Book {
    private String title;
    private double price;
    private static int totalNumBooks;
    private String author;
    public int numSold;
    public static final double MIN_PRICE = 2.0;

    public Book(String aTitle, double aPrice, String anAuthor) { ... }

    public double getPrice( ) { ... }

    private void lowerPrice( ) { ... }

    private static double newPrice(double oldPrice) { ... }
}
```

**Part (b)** Assume that the following (nonsense) code is in a method that is in the `Book` class. Find and fix (or cross out) all of the code that would cause compile-time errors.

```
Book oneBook = new Book("Harry Potter", 19.95, "Rowling");
MIN_PRICE = oneBook.getPrice();
oneBook.lowerPrice();
double price = Book.getPrice();
System.out.println(oneBook.numSold);
System.out.println(MIN_PRICE + Book.newPrice(5.75));
```

**Part (c)** Now assume that the code given above is *not* in the `Book` class. What new errors arise?

## Exercise 2: Multiple Identities (Aliasing and Equality)

For a person, an *alias* is a second name by which they are known. Superman's alias is Clark Kent. Wonder Woman's alias is Diana Prince. And Lester M. Gillis was a notorious criminal from Chicago who was known as Baby Face Nelson. Do you know any other people or fictional characters who had aliases?

In programming, two references are aliases if they refer to the same memory location. We saw some examples of aliasing last week when we acted out the Car class code. For this week, we'll use the Hideout and Bandit classes below.

```
public class Hideout {
    private String location;
    private boolean isSafe;

    // CONSTRUCTOR
    public Hideout(String loc) {
        location = loc;
        isSafe = true;
    }

    // GET AND SET METHODS
    public boolean getSafety() {
        return isSafe;
    }

    public void makeSafe() {
        isSafe = true;
    }

    public void makeUnsafe() {
        isSafe = false;
    }

    // EQUALS METHOD
    public boolean equals(Object ob) {
        Hideout other = (Hideout)ob;
        return isSafe == other.isSafe;
    }
}
```

```
public class Bandit {
    private String myName;
    private Hideout myHideout;

    // CONSTRUCTOR
    public Bandit(String name, Hideout hide) {
        myName = name;
        myHideout = hide;
    }

    // GET AND SET METHODS
    public Hideout getHideout() {
        return myHideout;
    }
}
```

### Part (a)

Trace the execution of the main method of the Test class below by drawing a memory diagram. Also record what is printed when the code executes.

```
1. public class Test {
2.     public static void main(String[] args) {
3.         Hideout h1 = new Hideout("CS Building");
4.         Hideout h2;
5.         Bandit b1, b2;
6.         b1 = new Bandit("Lucky Liz", h1);
7.         h2 = b1.getHideout();
8.         b2 = new Bandit("Slow Joe", h2);
9.         if (b2.getHideout().getSafety())
10.            System.out.println("b2's hideout is safe");
11.        else
12.            System.out.println("b2's hideout is NOT safe");
13.        h1.makeUnsafe();
14.        if (b2.getHideout().getSafety())
15.            System.out.println("b2's hideout is safe");
16.        else
17.            System.out.println("b2's hideout is NOT safe");
18.        h2 = new Hideout("CS Building");
19.        h2.makeUnsafe();
20.        h1 = h2;
21.        h1.makeSafe();
22.    }
23. }
```

### Part (b)

Where in the main method of the Test class would the expression `h1.equals(h2)` evaluate to true?

Where in the main method of the Test class would the expression `h1 == h2` evaluate to true?

### **Exercise 3: Designing a Class (static vs non-static fields and methods)**

For this exercise, we will design a `Student` class. Choose the fields and methods of the class based on the information given below. For each field and each method, choose an appropriate name and type, decide whether it should be static or not, and decide whether it should be final or not. Then write the class definition.

- Every student has a first name, a last name, a GPA (a number between 0.0 and 4.0), and a 4-digit ID number. The first student's ID number will be 1000, the next student's ID number will be 1001, then 1002, and so on.
- A `Student` object can be created given first and last names only (in which case the GPA is initialized to 0.0), or given first and last names and a GPA. In both cases, the student's ID is initialized to be the next available ID.
- A student's names and ID can be accessed, but cannot be modified. The GPA can be both accessed and modified.
- It should be possible to compare two `Student` objects to see if they are the same person. Two students with the same ID are considered equal.
- It should be possible to find out how many `Student` objects have been created so far.

## Exercise 4: Logical Thinking

Today's logical-thinking exercise is an old chess puzzle. The board (shown below) is a 3x3 part of the chessboard. The goal is to swap the red and blue knights, using a sequence of legal moves (of course, two knights may not occupy the same square at any time).

**B**: blue knight

**R**: red knight

<b>B</b>		<b>B</b>
<b>R</b>		<b>R</b>

**Problem:** See who can find the *shortest* sequence of moves to swap the knights. Can you find a good argument that this is the minimum number of moves required to solve the puzzle?

If you can solve the puzzle, consider the following additional question: Is it possible starting with the original configuration to arrange the knights as shown below? If not, why not?

<b>R</b>		<b>B</b>
<b>B</b>		<b>R</b>
