

## WES-CS GROUP MEETING #5

### Exercise 1: Matching Uses of Variables to their Declarations

Take a look at the *Book* class (similar to the one we used last week) and the *TestBook* class defined below.

```
public class Book {
    private String title;
    public double price;
    private int numSold;

    public Book(String aTitle, double aPrice) {
        title = aTitle;
        price = aPrice;
        this.numSold = 0;
    }

    public void setPrice(double price) {
        this.price = price;
    }

    public double salePrice(int percentOff) {
        double price;
        double discount = percentOff * .01;
        price = this.price;
        price -= price * discount;
        return price;
    }
}

public class TestBook {
    public static void main(String[] args) {
        Book myBook;
        double price;

        myBook = new Book("Winnie the Pooh", 9.75);
        price = myBook.salePrice(10);
        myBook.setPrice(price);
        System.out.println("new price is: " + myBook.price);
    }
}
```

## Part (a)

One way to think about what happens when a constructor or method is called is that a copy of the code is created, along with space for the parameters and local variables. Then the code is executed line by line. Remember that constructors and *instance* methods (non-static methods) also have a “secret” parameter called *this*, which is a pointer to the object being constructed (in the case of a constructor), or the object whose method was called.

What happens when the *main* method of the *TestBook* class executes? Find out by acting it out. Two people will be the *myBook* and *price* variables declared in the *main* method, three people will manage the three calls (to the *Book* constructor, and to methods *salePrice* and *setPrice*).

## Part (b)

Now look at the classes, *Practice* and *TestPractice*, defined below. Both classes declare and use the name *waldo*; sometimes incorrectly.

```
public class Practice {
    public int waldo;

    public Practice(int init) {
        waldo = init;
    }

    public void changeWaldo1(int val) {
        int waldo;

        waldo = val;
        this.waldo = waldo;
    }

    public int changeWaldo2(int val, int waldo) {
        this.waldo = val * waldo;
        return waldo;
    }
}

public class TestPractice {
    public static void main(String[] args) {
        Practice tst = new Practice(0);
        waldo = 100;
        tst.changeWaldo1(200);
        System.out.println( tst.changeWaldo2(10, 10) );
        tst.waldo = 50;
        this.waldo = 35;
    }
}
```

Start by finding and circling each occurrence of the name *waldo* in the code. Next, match each *use* of *waldo* with the corresponding declaration by drawing an arrow from the use to the declaration. If there is no matching declaration (in other words, if that use of *waldo* would cause a compile-time error), cross out that use of *waldo*.

Finally, eliminate the lines with errors, then trace the program execution.

## Exercise 2: Boolean conditions

Play a game using the decks of yellow and green cards. Each green card has an English sentence that is either true or false. Each yellow card has one of three symbols: !, &&, or || (meaning *not*, *and*, or). The game is played as follows:

- Each player starts with 6 cards, 3 yellow and 3 green.
- When it's your turn, if you can make a logical formula that evaluates to true using at least 4 of your cards, put down the formula and draw new cards of the same colors as the ones you used. Otherwise, trade in two of your cards for two new cards of the same color.
- The first person to put down at least 13 cards wins.

The precedences of the logical operators are as follows:

!	highest precedence
&&	next highest precedence
	lowest precedence

### Exercise 3: UML Diagrams

Below are new (incomplete) versions of the *Book* and *TestBook* classes.

```
public class Book {
    private String title;
    private Person author;
    private double price;

    public Book(String aTitle, Person anAuthor, double aPrice) {
        title = aTitle;
        price = aPrice;
        author = anAuthor;
    }

    public double getPrice( ) { ... }
    public void setPrice(double price) { ... }
    public String getTitle( ) { ... }
    public Person getAuthor( ) { ... }
    public void putOnSale( ) { ... }
    private void lowerPrice( ) { ... }
    private static double newPrice(double oldPrice, int discount) { ... }
}
```

```
public class TestBook {
    public static final int MONTH=12, DAY=16, YEAR=1775;

    public static void main(String[] args) {
        Book myBook;
        Person author;
        Date dob;

        dob = new Date(MONTH, DAY, YEAR);
        author = new Person("Jane", "Austen", dob);
        myBook = new Book("Pride and Prejudice", author, 20.50);
    }
}
```

On the next page is an incomplete UML diagram for the *Book*, *TestBook*, *Date*, and *Person* classes. Your job is to do the following:

- Fill in the boxes for the *Book* and *TestBook* classes.
- Write the definitions of the *Date* and *Person* classes so that your code is consistent with the UML diagram.

<b>Book</b>

<b>TestBook</b>

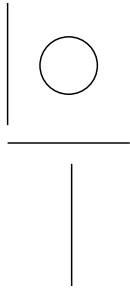
<b>Person</b>
<ul style="list-style-type: none"> <li>- <b>firstName: String</b></li> <li>- <b>lastName: String</b></li> <li>- <b>birthDate: Date</b></li> </ul>
<ul style="list-style-type: none"> <li>+ <b>Person(String, String, Date)</b></li> <li>+ <b>getFirstName(): String</b></li> <li>+ <b>getLastName(): String</b></li> <li>+ <b>getBirthDate(): String</b></li> </ul>

<b>Date</b>
<ul style="list-style-type: none"> <li>- <b>month: int</b></li> <li>- <b>day: int</b></li> <li>- <b>year: int</b></li> <li>+ <b><u>currYear</u>: int</b></li> </ul>
<ul style="list-style-type: none"> <li>+ <b>Date(int, int, int)</b></li> <li>+ <b>getDay(): int</b></li> <li>+ <b>getMonth(): int</b></li> <li>+ <b>getYear(): int</b></li> </ul>

*Space to write the Date and Person classes*

## Exercise 4: Logical Thinking

**Part (a):** Use 4 toothpicks and a Hershey's kiss to make the following picture of a cocktail glass with an olive in it:



Now move only two of the toothpicks to change the picture to a glass (the same shaped glass) with the olive outside the glass.

**Part (b):** Create an equilateral triangle using three toothpicks. Now add three more equilateral triangles of the same size as the original using only three more toothpicks.

Green cards

The Earth is flat

Madison is the capital of Wisconsin

In Java, \* has higher precedence than +

In Java, + has higher precedence than -

The moon is made of green cheese

17 is a prime number

There is not a DDR machine in Memorial Union

The dorms get Cartoon Network

Red and blue make purple

A chestnut tree makes acorns

Green cards

A giraffe has a black tongue

Ducks can't fly

We are learning C++ in CS302

To compare two Java strings, use `.equals` not `==`

The sun moves around the Earth

The UW-Madison was founded in 1492

There are 10 WES-CS groups

This room is CS 1240

Miss Muffet is afraid of spiders

## Green cards

The time now is 3:33pm

The sun sets in the west

Java programs start executing in a class named Main

The following code will compile: `boolean b = "true";`

An if statement will repeat until the condition is false

Bascom Hall is exactly one mile from the state capitol building.

`System.out.println()` is a static method.

Booleans can have three possible values.

Boolean algebra was invented by Bobby Boolean.

Green cards

Maine is the only state whose name is just one syllable.

It is snowing outside.

The University of Wisconsin was founded in 1492.

Wisconsin became a state in 1911.

The mascot of the University of Wisconsin is Bucky the Bobcat.

Pascal, Python, and Scheme are all names of programming languages.

Yellow cards

&&

&&

&&

&&

&&

&&

&&

&&

&&

&&

&&

&&

**Yellow cards**

||

||

||

||

||

||

||

||

||

||

||

||

!

!

!

**Yellow cards**

!

!

!

!

!

!

!

!

!

**Yellow cards**

(

(

(

(

(

(

(

(

(

(

(

(

(

(

(