

WES-CS GROUP MEETING #3

Exercise 1: Syntax

For this exercise, divide into groups of 2 or 3. Each group will get one set of cards: green, yellow, and pink, or buff, blue, and gold. The green/buff ones have an English description of some kind of Java code, the yellow/blue ones have a definition of the syntax for that code, and the pink/gold ones have an example of that kind of code. Your job is to match them up: find groups of three cards (one of each color) that go together.

Exercise 2: Writing Algorithms

Last week we assumed that we had a *Word* class that included the following methods:

void swap(int pos1, int pos2)

Swap the letter in position *pos1* with the letter in position *pos2*.

void moveToFront(int pos)

Move the letter in position *pos* to the beginning of the sequence (i.e., to position zero).

void moveToEnd(int pos)

Move the letter in position *pos* to the end of the sequence.

Part (a).

Below is an incomplete algorithm for reversing the letters in a word (for example, turning STOP into POTS, or HELLO into OLLEH) using only the *swap* method.

Step 1: Let *j* equal 0 and let *k* equal the number of letters in the word - 1.

Step 2: If _____ stop.

Step 3: word.swap(*j*, *k*)

Step 4: Let *j* equal _____ and let *k* equal _____.

Step 5: Goto Step 2.

Your job is to fill in the blanks.

Part (b).

Now assume we only have the *moveToFront* method. Write an algorithm for reversing the letters in a word.

If you finish early, try it again assuming that you only have the *moveToEnd* method.

Exercise 3: The Car Class

This exercise will help you to understand what happens when objects are declared and created, and when methods are called.

It will also help you to understand the difference between copying from one variable to another when the variable is an object, and when it is a primitive type (int, double, boolean, etc), as well as the difference between changing the values of variables that are objects vs primitive types.

First, take a look at the Car class defined on the next page.

Now execute the following code fragment; let one person play the role of each variable (myCar, yourCar, anotherCar, oldSpeed, and newSpeed). When a variable is assigned to, or one of its methods is called, the person playing the role of that variable should act out effects of the assignment or call.

```
Car myCar, yourCar, anotherCar;
int oldSpeed, currSpeed;

myCar = new Car("beep");
yourCar = new Car("honk");
anotherCar = myCar;

currSpeed = myCar.getCurrSpeed();
yourCar.changeSpeed(7);
anotherCar.changeSpeed(20);
currSpeed = myCar.getCurrSpeed();

myCar.blowHorn(2);
yourCar.blowHorn(3);
anotherCar.blowHorn(4);

oldSpeed = currSpeed;
myCar = yourCar;
currSpeed = myCar.getCurrSpeed();

myCar.changeSound("ooga");
myCar.blowHorn(currSpeed/5);
yourCar.blowHorn( yourCar.getCurrSpeed()/2 );
anotherCar.blowHorn( myCar.getCurrSpeed()/10 );
anotherCar = myCar;
```

```

class Car {
    /*******
     * data members
     *****/
    private int currSpeed;
    private String hornSound;

    /*******
     * public methods
     *****/
    /* constructor */
    public Car(String sound) {
        currSpeed = 0;
        hornSound = sound;
    }

    /* changeSound: change the horn sound */
    public void changeSound(String newSound) {
        hornSound = newSound;
    }

    /* blowHorn: blow the horn;
     * parameter numTimes tells you how many times
     */
    public void blowHorn(int numTimes) {
        while (numTimes > 0) {
            System.out.println(hornSound);
            numTimes--;
        }
    }

    /* changeSpeed: change speed */
    public void changeSpeed(int milesPerHour) {
        currSpeed = currSpeed + milesPerHour;
    }

    /* getCurrSpeed: return the current speed */
    public int getCurrSpeed() {
        return currSpeed;
    }
}

```

Exercise 4: Java variables and types

Remember that in Java

- variables must be declared and initialized before they are used,
- operators (like +, -, and so on) must be applied only to variables and literals of the correct type,
- in general, the types of the left and right-hand sides of an assignment must match (an exception is that an int value can be assigned to a long or double variable, but not vice-versa),
- the types of the arguments in a method call must match the types of the corresponding parameters, and the value returned must match the method's return type (again, an int can be used where a double is expected, but not vice-versa).

Keeping all this in mind, find all of the errors that the Java compiler would report in the following code.

```
public int doDivision( double denom ) {
    int returnVal;
    double num1 = 4.4;
    num2 = 5.5;
    returnVal = (num1 + num2)/denom;
    return returnVal;
}

public int computeVal( int oneVal ) {
    int returnVal;
    Integer bigInt1, bigInt2, bigInt3;
    bigInt1 = 222222222;
    bigInt2 = num1;
    bigInt3 = doDivision( "hello" );
    returnVal = (bigInt1 + bigInt2)/10000.0;
    Return returnVal;
}
```

Exercise 5: Logical Thinking

Assume that you have 8 coins, and you know that 7 are OK but one is bad. You know that the bad coin has a different weight than the good coins, but you don't know whether it's heavier or lighter.

Figure out how, using only a balance scale, you can find out which is the bad coin using just 3 weighings. Hint: Find a way to determine that half of the coins are OK with just 1 weighing.

Now do the same thing assuming that you have 9 coins, one of which is bad. (Still use just 3 weighings to find the bad coin.)

And now for a real challenge, do the same thing assuming that you have 13 coins.

GREEN OR BUFF CARDS

variable declaration (no initialization)

variable declaration with initialization

object declaration (no object creation)

object declaration and creation

class declaration

(non-constructor) method declaration

constructor declaration

method call

GREEN OR BUFF CARDS

assignment statement

field (instance variable, data member) declaration

print to the terminal

comment

YELLOW OR BLUE CARDS

<type> <var name>;

<type> <var name> = <expression>;

<class name> <object name>;

<class name> <object name> =
new <class name> (<args>);

<access specifier> class <class name> {
 <class member declarations>
}

<access specifier> <return type> <method name>(<params>) {
 <method body>
}

YELLOW OR BLUE CARDS

```
<access specifier> <class name>( <params> ) {  
    <method body>  
}
```

```
<object name>.<method name>( <args> );
```

```
<var name> = <expression>;
```

```
<access specifier> <type> <name> ;
```

```
System.out.print( <exp> );    or    System.out.println( <exp> );
```

```
// <text>           or           /* <text> */
```

PINK OR GOLD CARDS

```
int x;
```

```
Artist picasso;
```

```
double d = 5.5;
```

```
String name = "Bozo";      or      Time t = new Time(12, 10);
```

```
public class Time {  
    data member declarations  
    method declarations  
}
```

```
/* this is the tricky part */      or      // constructor
```

```
public void drawLineLeft( int len ) {  
    statements  
}
```

PINK OR GOLD CARDS

```
public Time( int hrs, int mins ) {  
    statements  
}
```

```
picasso.drawLineLeft( 10 );
```

```
picasso = new Artist();
```

```
System.out.print("hello");    or    System.out.println(x);
```

```
private int hours;
```