

# WES-CS GROUP MEETING #4

## Exercise 1: 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;
}

public boolean doLogic( ) {
    boolean bool1 = true;
    boolean bool2 = False;
    return (bool1 + bool2);
}
```

## Exercise 2: The Car Class

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

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).

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, oldSpeed, and newSpeed), and let another person be in charge of writing the output values on the board.

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

myCar = new Car("beep");
currSpeed = myCar.getCurrSpeed();
while (currSpeed < 10) {
    oldSpeed = currSpeed;    // copy from one int to another
    myCar.changeSpeed(7);
    currSpeed = myCar.getCurrSpeed();
    if ((currSpeed % 2) == 0) {
        myCar.blowHorn(currSpeed/5);
    } else {
        System.out.println("old speed: " + oldSpeed);
    }
}

yourCar = myCar; // copy from one Object to another
yourCar.changeSound("ooga");
yourCar.blowHorn( yourCar.getCurrSpeed()/7 );
myCar.blowHorn(1); // what happens here??
```

```

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! */
    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 3: The String class

The String class has many useful methods including the following:

String substr(int beginIndex, int endIndex)

Returns a new string that is the substring of this string that starts at *beginIndex* and ends at index *endIndex* - 1 (the index of the first character is 0). Error if *endIndex* is greater than the length of this string.

int indexOf(int ch)

Returns the index within this string of the first occurrence of *ch*. If no such character occurs in this string, then -1 is returned.

int compareTo(String anotherString)

Returns 0 if this string is the same as *anotherString*; returns a negative integer if this string comes before *anotherString* in lexicographic order (dictionary order); returns a positive integer if this string comes after *anotherString* in lexicographic order.

boolean equals(Object anObject)

Returns true if *anObject* is a String that represents the same sequence of characters as this string; otherwise returns false.

To get some practice using the String methods, play the following game (in pairs). (The cards to use are on the last four pages; they need to be cut up into individual cards.)

- Each person thinks of a word, 3 to 5 letters long. Whoever guesses their opponent's word first wins!
- Each person takes 2 cards.
- Alternate turns. When it's your turn, pick one card, then play one of your 3 cards. If it has a blank (e.g., *s.indexOf(\_\_)*), you get to fill in the blank with whatever you want. Your opponent tells you what value their word, *s*, would return if the method call on your card were made. For example, if your opponent is thinking of the word "hat" and you play *s.substr(1, 2)*, your opponent must say "a"; if you play *s.substr(2, 4)*, your opponent must say "error" (because "hat" has only 3 letters).
- You win if you play an *s.equals* card and your opponent says "true", or if you play an *s.compareTo* card and your opponent says 0 (i.e., you guessed their word).

## Exercise 4: Class methods and data vs Instance methods and data

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

Part (a)

Look at the Book class defined below. Say which data members are *class* data members and which are *instance* data members. Do the same for the methods.

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

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

    public double getPrice( ) {
        ...
    }

    private void lowerPrice( ) {
        ...
    }

    private static double newPrice(double oldPrice, int discount) {
        ...
    }
}
```

Part (b)

Assume that the following (nonsense) code is in class BookMain (*not* in the Book class) and find all of the errors.

```
Book oneBook = new Book("Harry Potter", 19.95, "Rowlings");
if (oneBook.getPrice > 10.00) {
    oneBook.lowerPrice();
}
double price = Book.getPrice();
System.out.println(oneBook.numSold);
System.out.println(MIN_PRICE);
```

Part (c)

Now assume that the code given above is in the Book class. Which are still errors, and how can you fix them?

s.length()	s.length()	s.length()
s.length()	s.length()	s.substr(0, 1)
s.substr(0, 1)	s.substr(0, 1)	s.substr(0, 1)
s.substr(0, 1)	s.substr(0, 2)	s.substr(0, 2)
s.substr(0, 2)	s.substr(0, 2)	s.substr(1, 2)
s.substr(1, 2)	s.substr(1, 2)	s.substr(1, 2)
s.substr(2, 3)	s.substr(2, 3)	s.substr(2, 3)
s.substr(2, 3)	s.substr(2, 4)	s.substr(2, 4)
s.substr(2, 4)	s.substr(2, 4)	s.substr(3, 4)

s.substr(3, 4)

s.substr(3, 4)

s.substr(3, 4)

s.substr(3, 5)

s.substr(3, 5)

s.substr(3, 5)

s.substr(3, 5)

s.indexOf('a')

s.indexOf('a')

s.indexOf('a')

s.indexOf('a')

s.indexOf('a')

s.indexOf('e')

s.indexOf('e')

s.indexOf('e')

s.indexOf('e')

s.indexOf('i')

s.indexOf('i')

s.indexOf('i')

s.indexOf('i')

s.indexOf('o')

s.indexOf('o')

s.indexOf('o')

s.indexOf('o')

s.indexOf('u')

s.indexOf('u')

s.indexOf('u')

s.indexOf( \_\_\_ )

s.indexOf( \_\_\_ )

s.indexOf( \_\_\_ )

s.indexOf( \_\_\_ )

s.indexOf( \_\_\_ )

s.indexOf( \_\_\_ )

s.indexOf( \_\_\_ )

s.indexOf( \_\_\_ )

s.indexOf( \_\_\_ )

s.indexOf( \_\_\_ )

s.indexOf( \_\_\_ )

s.indexOf( \_\_\_ )

s.indexOf( \_\_\_ )

s.indexOf( \_\_\_ )

s.indexOf( \_\_\_ )

s.indexOf( \_\_\_ )

s.indexOf( \_\_\_ )

s.indexOf( \_\_\_ )

s.indexOf( \_\_\_ )

s.indexOf( \_\_\_ )

s.indexOf( \_\_\_ )

s.compareTo( \_\_\_ )

s.compareTo( \_\_\_ )

s.compareTo( \_\_\_ )

s.compareTo( \_\_\_ )    s.compareTo( \_\_\_ )    s.compareTo( \_\_\_ )

s.compareTo( \_\_\_ )    s.compareTo( \_\_\_ )    s.compareTo( \_\_\_ )

s.compareTo( \_\_\_ )    s.compareTo( \_\_\_ )    s.compareTo( \_\_\_ )

s.compareTo( \_\_\_ )    s.compareTo( \_\_\_ )    s.compareTo( \_\_\_ )

s.compareTo( \_\_\_ )    s.compareTo( \_\_\_ )    s.compareTo( \_\_\_ )

s.equals( \_\_\_ )            s.equals( \_\_\_ )            s.equals( \_\_\_ )

s.equals( \_\_\_ )            s.equals( \_\_\_ )            s.equals( \_\_\_ )

s.equals( \_\_\_ )            s.equals( \_\_\_ )            s.equals( \_\_\_ )