

## WES-CS GROUP MEETING #6

### Exercise 1: Find Me a Formula (Java Arithmetic)

Translate the following formulas to Java expressions, using methods from the `Math` class where appropriate. Assume that all variables are of type `double`.

- (a)  $\frac{n(n+1)(2n+1)}{6}$
- (b)  $2 \cos^2(a) - 1$
- (c)  $\frac{-b + \sqrt{b^2 - 4ac}}{2a}$
- (d)  $vt + 1/2 at^2$
- (e)  $|x + n^{3/2} - \tan^5(\min(a, b))|$

#### Some Math Class Methods for Double Values

<code>static double</code>	<code>abs(double a)</code> Returns the absolute value of a double value.
<code>static double</code>	<code>cbrt(double a)</code> Returns the cube root of a double value.
<code>static double</code>	<code>cos(double a)</code> Returns the trigonometric cosine of an angle.
<code>static double</code>	<code>max(double a, double b)</code> Returns the greater of two double values.
<code>static double</code>	<code>min(double a, double b)</code> Returns the smaller of two double values.
<code>static double</code>	<code>pow(double a, double b)</code> Returns the value of the first argument raised to the power of the second argument.
<code>static double</code>	<code>sin(double a)</code> Returns the trigonometric sine of an angle.
<code>static double</code>	<code>sqrt(double a)</code> Returns the correctly rounded positive square root of a double value.
<code>static double</code>	<code>tan(double a)</code> Returns the trigonometric tangent of an angle.

## Exercise 2: If and Switch Statements

### Part (a).

Divide into 3 groups. Each group choose one set of cards: yellow and pink, or blue and green, or gold and white.

Each yellow/blue/gold card has a code fragment that includes an *if* statement, and each pink/green/white card has a code fragment that includes a *switch* statement. Your job is to match each *if*-fragment with the *best* equivalent *switch*-fragment.

If there is more than one equivalent *switch*-fragment, the *best* one is the simplest (the one you'd prefer to see in an actual program).

### Part (b).

You should have some *switch*-fragments that are similar but not equivalent to any of the *if*-fragments. For each of these, explain why the *switch*-fragment isn't equivalent to the similar *if*-fragment.

### Exercise 3: Nick the Sub (Algorithms)

#### Part (a).

For this exercise, we'll say that a nickname is formed by taking any non-empty prefix of a name (the first one or more letters) followed by "ie". For example, this rule gets us the following:

Name	Nickname	Not a Nickname
Susan	Susie	Suzy
Martin	Martie	Smartie
Ann	Annie	nnie

It also gets us some nonsense nicknames, which we won't worry about:

Name	Nickname
Susan	Sie
Martin	Martinie
Ann	Anie

Your job is to write an algorithm (in English or as a Java method) to determine whether one string (called `nick`) is a nickname for another string (called `name`). For example, if `nick` is "Susie" and `name` is "Susan" then your algorithm should say that `nick` **is** a nickname for `name` (or your Java code should return `true`). If `nick` is "Susie" and `name` is "George" then your algorithm should say that `nick` is **not** a nickname for `name`.

Whether you write your algorithm in English or in Java, you should make use of the `String` methods to find out how long `nick` and `name` are, and to extract and compare substrings. Hint: Think about which pieces of the two strings you need to look at, and what the values of those pieces need to be.

Some useful `String` methods are given on the next page.

## Some String Methods

char	<code>charAt(int index)</code> Returns the char value at the specified index.
boolean	<code>equals(Object anObject)</code> Compares this string to the specified object.
int	<code>indexOf(int ch)</code> Returns the index within this string of the first occurrence of the specified character.
int	<code>indexOf(int ch, int fromIndex)</code> Returns the index within this string of the first occurrence of the specified character, starting the search at the specified index.
int	<code>indexOf(String str)</code> Returns the index within this string of the first occurrence of the specified substring.
int	<code>indexOf(String str, int fromIndex)</code> Returns the index within this string of the first occurrence of the specified substring, starting the search at the specified index.
int	<code>length()</code> Returns the length of this string.
String	<code>substring(int beginIndex)</code> Returns a new string that is the substring of this string that starts at position <code>beginIndex</code> .
String	<code>substring(int beginIndex, int pastEnd)</code> Returns a new string that is a substring of this string that starts at position <code>beginIndex</code> and ends at position <code>pastEnd-1</code> .

**Part (b).**

Now we'll write an algorithm that figures out whether the string in `word1` is a **subsequence** of the string in `word2`. String `word1` is a subsequence of `word2` if all of the letters in `word1` occur in `word2` in the same order, but perhaps with some other letters in between. For example

<code>word1</code>	<code>word2</code>	<code>word1</code> is a subsequence of <code>word2</code> ?
<code>win</code>	<code>wisconsin</code>	yes
<code>wise</code>	<code>wisconsin</code>	no
<code>snow</code>	<code>wisconsin</code>	no
<code>sin</code>	<code>wisconsin</code>	yes

Again, you can write your algorithm in English or in Java, but in either case you should use the `String` methods.

**Exercise 4: Does Anybody Really Know What Time it is? (Loops)**

**Part (a)**

Write code that prints each minute between 1:00pm and 2:59pm using one or more loops; i.e., your code should print:

```
1:00pm
1:01pm
...
1:59pm
2:00pm
2:01pm
...
2:59pm
```

Swap your code with your neighbor and see if either of you can find any errors. Once you think you both have correct code, you can test it by typing it in, compiling it, and seeing if it works.

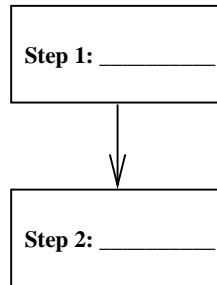
**Part (b)**

Write code to print all of the 15-minute interval times between 11:00am and 2:45pm (i.e., 11:00am, 11:15am, ..., 2:00pm, 2:45pm). Note that noon is 12:00pm.

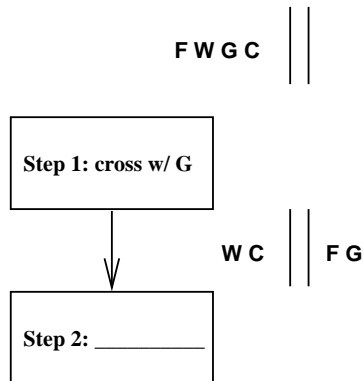
## Exercise 5: To Market to Market (Logical Thinking)

A farmer is taking a wolf, a goat, and a head of cabbage to the market. To do this, the farmer must cross a river. He has a boat, but it is only big enough to carry him and at most one of the wolf, goat, or cabbage (the farmer can also choose to cross the river alone). If the farmer leaves the wolf and the goat alone together, the wolf will eat the goat. If the farmer leaves the goat and the cabbage alone together, the goat will eat the cabbage.

Work with a partner to find an algorithm that gets the farmer, the wolf, the goat, and the cabbage across the river without anything getting eaten. Write a flowchart for the algorithm that looks like this:



where each step says who the farmer takes across the river. You may find it helpful to keep track of who is on which side of the river by drawing *state diagrams* before and after each step of the algorithm as shown below.



There are two solutions to this problem. See if two groups found the two solutions; if not, everyone try to find the second solution.

### Yellow or Blue or Gold Cards

```
if (x == 0) y = 1;  
else if (x == 1) y = 2;  
else y = 3;
```

```
if ( x == 4 ) {  
    x += 4;  
} else if ( (x > 4) && (x <= 7) ) {  
    x += 7;  
}
```

```
if ( x == 1 || x == 3 ) {  
    System.out.println("Good Morning!");  
} else {  
    System.out.println("Good Afternoon!");  
}
```

```
if (x == 0 && y == 0) doSomething();  
else if (x == 0 && y == 1) doAnotherThing();  
else doSomethingElse();
```

```
if ( x > 2 && x < 4 ) x = 10;  
if ( x <= 0 || x > 4 ) x = 100;
```

## Pink or Green or White Cards

```
switch (x) {  
    case 0: y = 1;  
           break;  
    case 1: y = 2;  
           break;  
    default: y = 3;  
}
```

```
y = 3;  
switch (x) {  
    case 0: y = 1;  
}  
switch (x) {  
    case 1: y = 2;  
}
```

```
switch (x) {  
    case 0: y = 1;  
    case 1: y = 2;  
    default: y = 3;  
}
```

### Pink or Green or White Cards

```
switch (x) {  
    case 4 : x = x + 4; break;  
    case 5 :  
    case 6 :  
    case 7 : x = x + 7; break;  
}
```

```
switch(x) {  
    case 5:  
    case 6:  
    case 7: x = x + 7;  
           break;  
    default: x = x + 4;  
}
```

```
switch(x) {  
    case 1:  
    case 3: System.out.println("Good Morning!");  
           break;  
    default: System.out.println("Good Afternoon!");  
}
```

### Pink or Green or White Cards

```
switch(x) {  
    case 1:  
    case 3: System.out.println("Good Morning!");  
}  
System.out.println("Good Afternoon!");
```

```
switch (x) {  
    case 0: switch (y) {  
        case 0: doSomething();  
            break;  
        case 1: doAnotherThing();  
            break;  
        default:doSomethingElse();  
    }  
    break;  
    default: doSomethingElse();  
}
```

### Pink or Green or White Cards

```
switch (x) {  
  case 0: switch (y) {  
    case 0: doSomething();  
           break;  
    case 1: doAnotherThing();  
  }  
  break;  
  default: doSomethingElse();  
}
```

```
switch (x) {  
  case 0: if (y==0) doSomething();  
         else doAnotherThing();  
         break;  
  default: doSomethingElse();  
}
```

## Pink or Green or White Cards

```
switch (x) {  
    case 1:  
    case 2:  
    case 4: break;  
    default : x = 100;  
}
```

```
switch (x) {  
    case 3: x = 10;  
}
```

```
switch (x) {  
    case 1:  
    case 2:  
    case 3:  
    case 4: break;  
    default: x = 100;  
}
```

### Pink or Green or White Cards

```
switch (x) {  
    case 1:  
    case 2:  
    case 4: break;  
    case 3: x = 10; break;  
    default : x = 100;  
}
```

```
switch (x) {  
    case 3: x = 10; break;  
    default : x = 100;  
}
```