WES-CS GROUP MEETING #4

Exercise 1: Arithmetic expressions

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

(a) $\dfrac{n\,(n+1)\,(2n+1)}{6}$

(b) $a^2 + b^2 + 2ab\,\cos(c)$

(c) $2\cos^2(a) - 1$

(d) $\dfrac{-b + \sqrt{b^2 - 4ac}}{2a}$

(e) $ut + 1/2\ at^2$

Some Math Class Methods for Double Values

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

Exercise 2: The String class

The String class has many useful methods including the following:

int length()
> Returns the number of characters in this string.

String substring(int beginIndex, int pastEnd)
> Returns a new string that is the substring of this string that starts at position *beginIndex* and ends at position *pastEnd* - 1 (the position of the first character is 0). Error if *beginIndex* is negative, or *pastEnd* is greater than the length of this string.

int indexOf(int ch)
> Returns the position 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).

- Each person thinks of a word, 3 to 5 letters long. Whoever guesses their opponent's word first wins!
- Each person takes 2 cards.
- If you ever have more then one of the same card, you may trade one of them in for a new card if you wish.
- Alternate turns. When it's your turn, pick up one new 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.substring(1, 2)*, your opponent must say "a"; if you play *s.substring(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 3: Algorithms

Assume that you have a *String* variable called *word*. Write an algorithm to determine whether the sequence of characters in *word* is a palindrome (i.e., is the same forward and backward). You can write your algorithm using "Step 1", "Step 2", etc, as we have done in the past, or you can write actual Java code. You might find it helpful to use the following *String* method (in addition to the ones on the previous page):

char charAt(int index)

     Returns the character at the specified index.

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

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;
   private 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, int discount) { ... }
}
```

Part (b) Assume that the following (nonsense) code is in a method that is in the *Book* class. Find and fix all of the errors.

```
     Book oneBook = new Book("Harry Potter", 19.95, "Rowlings");
     MIN_PRICE = oneBook.getPrice();
     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 *not* in the *Book* class. What new errors arise?

Exercise 5: Logical Thinking

A person visiting an island with a nasty dictator was thrown in jail for no reason, and told that he had been sentenced to death. However, he would be able to choose the method of execution in the following unusual way: He was to make one statement of fact (e.g. "I am a bird" or "I am wearing a brown, polyester shirt"). If the statement is true, he will be beheaded. If it is false, he will be hanged. At this point, it is clear that the dictator is not a logician, because he has given the prisoner a way out! What could he say in order to avoid execution?

The prisoner manages to solve the question above (did you?), and he languishes in jail for a while. Then one day he manages to escape, and he flees down the road, trying to reach the sea. He gets to a fork in the road, and there is a native of the island standing there. Now there is something strange about this island's natives: each person either always tells the truth, or always lies. The prisoner doesn't know whether the native is a liar or a truth-teller. What single yes-no question could he ask so that he would know which fork in the road leads to the sea, no matter whether the native is a liar or a truth-teller?

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

s.substring(3, 4)	s.substring(3, 4)	s.substring(3, 4)
s.substring(3, 5)	s.substring(3, 5)	s.substring(3, 5)
s.substring(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(___)

s.length( )   s.length( )   s.length( )

s.length( )   s.length( )   s.length( )

s.length( )   s.length( )   s.length( )

s.length( )   s.length( )   s.length( )

s.equals(___)   s.equals(___)   s.equals(___)

s.equals(___)   s.equals(___)   s.equals(___)

s.equals(___)   s.equals(___)   s.equals(___)

s.equals(___)   s.equals(___)   s.equals(___)