

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)
$$\frac{n(n+1)(2n+1)}{6}$$

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

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

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

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

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 substr(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 *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.
- 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 3: Algorithms

Part (a).

Write Java code that figures out how to give change for an amount that is no less than 1 cent and no greater than 99 cents, using the fewest possible coins. For example, if the amount is 69 cents, you would use 2 quarters, 1 dime, 1 nickle and 4 pennies.

Complete the code segment started below, so that when the code finishes executing, variables *numQuarters*, *numDimes*, *numNickles*, and *numPennies* have the right values. (Assume that there is a variable named *amount* that has been declared to be an *int*, and that its value is between 1 and 99.)

```
int numQuarters, numDimes, numNickles, numPennies;  
numQuarters =
```

Part (b).

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.

```
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 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

1. Meet yourself on the mountain

A pilgrim starts walking up a narrow mountain path at 6am. The pilgrim walks at different speeds, stopping to eat, sometimes going back a few steps to look at a flower, but never leaving the path. The pilgrim eventually arrives at the top at 10pm the same day. The pilgrim camps out over night and starts down the same path at 6am the next day. After stopping to eat and so forth the pilgrim arrives at the bottom at 10pm. Is there some point on the path that the pilgrim occupied at exactly the same time on the two different days?

2. True or false?

A person was imprisoned on a very strange island: On that island, each person either always told the truth, or always lied. One day the prisoner escaped, and fled down the road, trying to reach the sea. The prisoner got to a fork in the road, and there was a native of the island standing there. The prisoner didn't know whether the native was a liar or a truth-teller. What single yes-no question could be asked so that the prisoner would know which fork in the road led to the sea, no matter whether the native was a liar or a truth-teller?

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(___)

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

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

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

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