

# WES-CS GROUP MEETING #10

## Exercise 1: Putting together a Java program

For this exercise, you must prepare some “cards” (strips of paper). Print pages 5–11 then cut them up into horizontal strips (cut in the gaps between lines), and mix up the strips. Each strip will have some Java code.

Now consider writing a Java program that reads in a simple equation like one of the following from a file:

```
3 + 4 = 7
53 - 13 = 40
1 * 5 = 23
4 / 6 = 0
```

then tests whether the equation is correct or not. You can do this by writing a class named `CheckEquation` with just a `main` method that performs each of the following tasks in turn:

- (1) Ask the user to enter the filename, then create a `BufferedReader` ready to read from the file.
- (2) Read a line of input into a `String` variable.
- (3) Use a `StringTokenizer` to break up the line into 5 parts (each stored in its own `String` variable): the first number, the operator, the second number, the equals sign, and the answer.
- (4) Convert the first, third, and fifth parts into integers.
- (5) Use the second part (the operator) to decide which arithmetic operation to perform, perform that operation, then compare your answer with the one you read in.

Break up into groups of two. Each group should choose one of the tasks. Implement your task by choosing the right pieces of code (the “cards” that you made by printing and cutting up pages). If you finish early, choose another task that hasn’t been handled yet. When all of the tasks are finished, put the code together to make a complete Java program.

## Exercise 2: Arrays

Print pages 12–15 on blue paper, and print pages 16–18 on pink paper. Cut each page into strips. The blue strips will have code fragments, each of which does something to an array called `arr` (in each case, assume that `arr` is a non-empty, initialized array of `int` values). The pink strips will have English descriptions of what the code fragments do. For each blue strip, find a matching pink strip.

## Exercise 3: The Geography Game

Two or more people can play the geography game. The first person says the name of a city (in any country in the world). The next person must say the name of a city that starts with the *last* letter of the previous city name. No city can be used more than once. If you can't think of a city name, you're out (and the last person in wins).

First, try playing the geography game a few times. Then we'll write a Java program that will play the game with you.

The Java program should work as follows:

- It should define one class named `Geog`.
- The `Geog` class should have a constructor that takes one `String` argument, a file name. The file should contain city names (all lower-case letters, one per line, with no spaces; e.g., "New York" would be in the file as "newyork"). The constructor should read the names from the file into an `ArrayList` that is a field of the `Geog` class.
- The `Geog` class should have a method named `playGame` that plays one game with you. It should ask you for the first city name, and then it should use its `ArrayList` of city names to respond to each name that you enter. It should also check the city names that you enter to make sure that they start with the right letter. If it can't respond to one of the names you enter, it should tell you that you win, and return. If you type in a name whose first letter doesn't match the last letter of the last city name, it should tell you that you lose, and return.
- The `Geog` class should have a main method that asks you for the name of the file that contains city names. Then it should use that name to create a `Geog` object, and then it should call that object's `playGame` method. When the method returns it means the game is over, and the program ends.

## Designing the Geography Program

Below are descriptions of some methods that you might want to include in your `Geog` class, as private “helper” methods for the game itself, or to help you test your code as you develop it. Which would actually be useful and why? Are there any other methods that you should implement? Decide in what order you should implement your methods, and write the program!

- (1) Given the name of a file containing city names, one per line, read all names into this `Geog` object's `ArrayList`.
- (2) Print all of the names in this `Geog` object's `ArrayList`.
- (3) Check whether one name is the same as another name backwards.
- (4) Remove a given name from this `Geog` object's `ArrayList` (if it is there).
- (5) Check whether the first letter in a name is a vowel.
- (6) Find the longest name in this `Geog` object's `ArrayList`.
- (7) Read a name typed in by the user of the program.
- (8) Check whether the last letter of one name is the same as the first letter of another name.
- (9) Check whether two names have the same length.
- (10) Find a name in this `Geog` object's `ArrayList` that starts with a given letter.
- (11) Find a name in this `Geog` object's `ArrayList` that ends with a given letter.

```
import java.util.*;
```

```
import java.io.*;
```

```
class CheckEquation {
```

```
    public static void main(String[] args) {
```

```
        BufferedReader userBr = new BufferedReader(new InputStreamReader( System.in ));
```

```
        System.out.print("enter file name: ");
```

```
        try {
```

```
String name;
```

```
BufferedReader fileBr;
```

```
String oneEquation;
```

```
name = userBr.readLine();
```

```
File inFile = new File(name);
```

```
fileBr = new BufferedReader(new FileReader(inFile));
```

```
oneEquation = fileBr.readLine();
```

```
StringTokenizer st = new StringTokenizer(oneEquation);
```

```
if (st.countTokens() != 5) {  
    System.err.println("Wrong number of parts in input equation");  
    System.exit(1);  
}
```

```
String num1 = st.nextToken();
```

```
String op = st.nextToken();
```

```
String num2 = st.nextToken();
```

```
String eqSign = st.nextToken();
```

```
String num3 = st.nextToken();
```

```
int firstNum, secondNum, answer;
```

```
Integer oneNum = new Integer(num1);
```

```
firstNum = oneNum.intValue();
```

```
oneNum = new Integer(num2);
```

```
secondNum = oneNum.intValue();
```

```
oneNum = new Integer(num3);
```

```
answer = oneNum.intValue();
```

```
int myAnswer = 0;
```

```
switch (op.charAt(0)) {
```

```
case '+':
```

```
    myAnswer = firstNum + secondNum;
```

```
    break;
```

```
case '-':
```

```
    myAnswer = firstNum - secondNum;
```

```
    break;
```

```
case '*':
```

```
    myAnswer = firstNum * secondNum;
```

```
    break;
```

```
case '/':
```

```
    myAnswer = firstNum / secondNum;
```

```
    break;
```

```
default:
```

```
    System.err.println("Bad operation in input equation");
```

```
    System.exit(1);
```

```
}
```

```
if (answer == myAnswer) {
```

```
    System.out.println("The equation is correct!");
```

```
} else {
```

```
    System.out.println("The equation is wrong!");
```

```
}
```

```
} catch (NumberFormatException ex) {
```

```
    System.err.println("Bad integer value in input equation");
```

```
} catch (FileNotFoundException ex) {  
    System.err.println("Error opening file");  
    System.exit(1);  
}
```

```
} catch (IOException ex) {  
    System.err.println("I/O Error");  
    System.exit(1);  
}
```

```
}
```

```
}
```

```
for (int k=0; k < arr.length/2; k++) {  
    int tmp = arr[k];  
    int index = arr.length-(k+1);  
    arr[k] = arr[index];  
    arr[index] = tmp;  
}
```

```
for (int k=0; k < arr.length-1; k++) {  
    if (arr[k] > arr[k+1]) return false;  
}  
return true;
```

```
for (int k=0; k < arr.length-1; k++) {  
    for (int j=k+1; j < arr.length; j++) {  
        if (arr[k] == arr[j]) return false;  
    }  
}  
return true;
```

```
for (int k=0; k < arr.length-1; k++) {  
    for (int j=k; j < arr.length; j++) {  
        if (arr[k] == arr[j]) return false;  
    }  
}  
return true;
```

```
for (int k=1; k < arr.length-1; k++) {  
    if (arr[0] == arr[k]) return false;  
}  
return true;
```

```
int tmp = 0;  
for (int k=0; k < arr.length; k++) {  
    tmp += arr[k];  
}  
return (tmp > 0);
```

```
for (int k=0; k < arr.length; k++) {  
    if (arr[k] >= N) return false;  
}  
return true;
```

```
int tmp = arr[0];  
for (int k=1; k < arr.length; k++) {  
    if (arr[k] > tmp) tmp = arr[k];  
}  
return tmp;
```

```
int tmp = 0;  
for (int k=0; k < arr.length; k++) {  
    tmp += arr[k];  
}  
return (double)tmp/arr.length;
```

```
int tmp = 0;
for (int k=1; k < arr.length; k++) {
    if (arr[k] == arr[0]) tmp++;
}
return tmp;
```

The code reverses the order of the values in the array.

The code returns true if the values in the array are in sorted order, from low to high.

The code returns true if the array contains no duplicate values (i.e., no value is stored in more than one place in the array).

The code always returns false.

The code returns true if the value in `arr[0]` doesn't occur in any other place in the array.

The code returns true if the sum of the values in the array is positive.

The code returns true if all values in the array are less than N.

The code returns the largest value in the array.

The code returns the average of the values in the array.

The code returns the number of times the value in `arr[0]` occurs in other places in the array, too.

The code returns true if the values in the array are in sorted order, from high to low.

The code always returns true.

The code returns true if all values in the array are positive.

The code returns true if the number of values in the array is less than N.

The code returns the smallest value in the array.

The code checks whether the values in the array are the same forwards and backwards.