Some practice with basic loop algorithms: step-by-step loops, sentinel loops, accumulation loops, and nested loops.

1. Basic Step-by-Step Loops

A basic step-by-step loop has an int variable with a start value, an end value, and an update value. The algorithm for this kind of loop looks like this:

```java
int currentStep = START_VALUE;
while(currentStep < END_VALUE) {
    // do something, which may involve the value of currentStep
    currentStep += UPDATE_VALUE;
}
```

(or, equivalently)

```java
for( int currentStep = START_VALUE;
    currentStep < END_VALUE;
    currentStep += UPDATE_VALUE)
{  
    // do something, which may involve the value of currentStep
}
```

a. Definite-length loops

In definite-length loops, the problem will tell you exactly how many times the loop should execute.
1. Print “I will not use a variable without declaring and initializing it first” 100 times on the screen.

First, copy the template loop from above. I will use the while loop template:

```java
int currentStep = START_VALUE;
while (currentStep < END_VALUE) {
    // do something, which may involve the value of currentStep
    currentStep += UPDATE_VALUE;
}
```

The way this is written right now, though, is not legal Java code. That’s because we’re using 3 variables (START_VALUE, END_VALUE, and UPDATE_VALUE) that have not been declared or initialized.

So the next step is to replace these with well-chosen values, so that this loop executes 100 times. I will pick START_VALUE = 1, END_VALUE = 100, and UPDATE_VALUE = 1, like this:

```java
int currentStep = 1; // replace START_VALUE with 1
while (currentStep < 100) { // replace END_VALUE with 100
    // do something, which may involve the value of currentStep
    currentStep += 1; // replace UPDATE_VALUE with 1
}
```

Finally, add a `System.out.println()` statement to the body of the loop, so that it gets executed each time through the loop.

```java
int currentStep = 1;
while (currentStep < 100) {
    System.out.println("I will not use a variable without declaring and initializing it first");
    currentStep += 1;
}
```
2. Create an array of 20 ints. Set the value of the first element to 1, the next element equal to 2, etc., so that the last element equals 20.

First, initialize an array variable to contain 20 ints, and copy the template from above:

```java
int [] myArray = new int[20];
int currentStep = START_VALUE;
while(currentStep < END_VALUE) {
    // do something, which may involve the value of currentStep
    currentStep += UPDATE_VALUE;
}
```

Next, decide on a START_VALUE, END_VALUE, and UPDATE_VALUE so that the currentStep variable goes through all the indices of myArray. That means, START_VALUE = 0, END_VALUE = 20, and UPDATE_VALUE = 1.

```java
int [] myArray = new int[20];
int currentStep = 0;
while(currentStep < 20) { // or, currentStep < myArray.length
    // do something, which may involve the value of currentStep
    currentStep += 1;
}
```

Finally, put an assignment statement in the body of the loop, so that the loop updates the cell at position currentStep each time through the loop.

```java
int [] myArray = new int[20];
int currentStep = 0;
while(currentStep < 20) { // or, currentStep < myArray.length
    myArray[currentStep] = currentStep;
    currentStep += 1;
}
```
3. Create an array of 20 Point objects. Initialize each Point object so that its x coordinate is a random number between 1 and 10, and its y coordinate is a random number between -5 and 5.

First, initialize an array variable to contain 20 Points, and copy the template from above:
```
Point [] myArray = new Point[20];
int currentStep = START_VALUE;
while(currentStep < END_VALUE) {
    // do something, which may involve the value of currentStep
    currentStep += UPDATE_VALUE;
}
```

Next, decide on a START_VALUE, END_VALUE, and UPDATE_VALUE so that the currentStep variable goes through all the indices of myArray. That means, START_VALUE = 0, END_VALUE = 20, and UPDATE_VALUE = 1.
```
Point [] myArray = new Point[20];
int currentStep = 0;
while(currentStep < 20) { // or, currentStep < myArray.length
    // do something, which may involve the value of currentStep
    currentStep += 1;
}
```

Finally, put an assignment statement in the body of the loop, so that the loop updates the cell at position currentStep each time through the loop. You’ll need a random number generator, which it’s best to declare and initialize outside the loop.
```
Random rand = new Random();
Point [] myArray = new Point[20];
int currentStep = 0;
while(currentStep < 20) { // or, currentStep < myArray.length
    int x = rand.nextInt(10)+1;
    int y = rand.nextInt(11)-5;
    myArray[currentStep] = new Point(x, y);
    currentStep += 1;
}
```
b. Indefinite-length loops

1. Read in an int N from the keyboard. Print all of the numbers between 1 and N.

First create a Scanner, read in N, and copy the loop template from above:

```java
Scanner input = new Scanner(System.in);
int N = input.nextInt();
int currentStep = START_VALUE;
while(currentStep < END_VALUE) {
    // do something, which may involve the value of currentStep
    currentStep += UPDATE_VALUE;
}
```

Next, decide on a value for START_VALUE, END_VALUE, and UPDATE_VALUE so that the loop executes N times. The easiest way is to make START_VALUE = 1, END_VALUE = N+1, and UPDATE_VALUE = 1.

```java
Scanner input = new Scanner(System.in);
int N = input.nextInt();
int currentStep = 1;
while(currentStep < N+1) {
    // or, equivalently, currentStep<=N
    // do something, which may involve the value of currentStep
    currentStep += 1;
}
```

Finally, add a println statement so that the loop prints the value of currentStep each time through:

```java
Scanner input = new Scanner(System.in);
int N = input.nextInt();
int currentStep = 1;
while(currentStep < N+1) {
    // or, equivalently, currentStep<=N
    System.out.println(currentStep);
    currentStep += 1;
}
```
2. Read in an int N from the keyboard. Create an array of N ints, and store the numbers between 1 and N in the array.

First create a Scanner, read in N, create an array, and copy the loop template from above:

```java
Scanner input = new Scanner(System.in);
int N = input.nextInt();
int [] myArray = new int[N]; // an array with N zeros
int currentStep = START_VALUE;
while(currentStep < END_VALUE) {
    // do something, which may involve the value of currentStep
    currentStep += UPDATE_VALUE;
}
```

Next, decide on a value for START_VALUE, END_VALUE, and UPDATE_VALUE so that the currentStep variable goes through each index of the array. The easiest way is to make

START_VALUE = 0, END_VALUE = N, and UPDATE_VALUE = 1.

```java
Scanner input = new Scanner(System.in);
int N = input.nextInt();
int [] myArray = new int[N]; // an array with N zeros
int currentStep = 0;
while(currentStep < N) {
    // do something, which may involve the value of currentStep
    currentStep += 1;
}
```

Finally, add an assignment statement, so that the loop sets the value of the cell at position currentStep each time through the loop:

```java
Scanner input = new Scanner(System.in);
int N = input.nextInt();
int [] myArray = new int[N]; // an array with N zeros
int currentStep = 0;
while(currentStep < N) {
    myArray[currentStep] = currentStep;
    currentStep += 1;
}
```
3. Generate a random number N between 1 and 100. Print every multiple of 7 between 1 and N (eg., if N = 31, it should print 7, 14, 21, and 28).

First create a random number generator, generate a number N, and copy the loop template from above:
```
Random rand = new Random();
// rand.nextInt(100) returns a number in the range [0, ..., 99].
// Add 1 to this to get a number between [1, ..., 100].
int N = rand.nextInt(100) + 1;
int currentStep = START_VALUE;
while(currentStep < END_VALUE) {
    // do something, which may involve the value of currentStep
    currentStep += UPDATE_VALUE;
}
```

Next, choose values for START_VALUE, END_VALUE, and UPDATE_VALUE so that the loop sets the currentStep to every multiple of 7. The first multiple of 7 between 1 and N is 7 itself, so START_VALUE should be 7. Every time after that, the next multiple of 7 will be 7 more than the previous one, so the UPDATE_VALUE should be 7. Since we don’t want to print anything bigger than N, the END_VALUE should be N.
```
Random rand = new Random();
int N = rand.nextInt(100) + 1;
int currentStep = 7;
while(currentStep < N) {
    // do something, which may involve the value of currentStep
    currentStep += 7;
}
```

Finally, add a println statement to the body of the loop (between the curly braces) to print out the value of currentStep each time through.
```
Random rand = new Random();
int N = rand.nextInt(100) + 1;
int currentStep = 7;
while(currentStep < N) {
    System.out.println(currentStep);
    currentStep += 7;
}
```
4. Generate a random number $N$ between 1 and 100. Read a character $C$ from the keyboard. Create a char array of length $N$, and initialize every element to the character stored in variable $C$. Construct a String from the char array.

First create a random number generator, generate a number $N$, create a Scanner, and read a character $C$:

```java
Random rand = new Random();
// rand.nextInt(100) returns a number in the range [0, ..., 99].
// Add 1 to this to get a number between [1, ..., 100].
int N = rand.nextInt(100) + 1;
Scanner input = new Scanner(System.in);
System.out.println("Please enter a letter: ");
// charAt(0) gets the first letter in what the user enters
char C = input.next().charAt(0);
```

Next, create a char array of length $N$, and copy the loop template from above:

```java
Random rand = new Random();
int N = rand.nextInt(100) + 1;
Scanner input = new Scanner(System.in);
System.out.println("Please enter a letter: ");
char C = input.next().charAt(0);
char[] myArr = new char[N];
int currentStep = 0;
while(currentStep < myArr.length) {
    // do something, which may involve the value of currentStep
    currentStep += 1;
}
```

Next, choose values for `START_VALUE`, `END_VALUE`, and `UPDATE_VALUE` so that `currentStep` goes through each possible index of the array. The easiest way is to change `START_VALUE` to 0, `END_VALUE` to $N$ (or `myArr.length`) and `UPDATE_VALUE` to 1.

```java
Random rand = new Random();
int N = rand.nextInt(100) + 1;
Scanner input = new Scanner(System.in);
System.out.println("Please enter a letter: ");
char C = input.next().charAt(0);
char[] myArr = new char[N];
int currentStep = 0;
while(currentStep < myArr.length) {
    // do something, which may involve the value of currentStep
    currentStep += 1;
}
```
Finally, add an assignment statement that sets the value of myArr[currentStep] each time through the loop.

```java
Random rand = new Random();
int N = rand.nextInt(100) + 1;
Scanner input = new Scanner(System.in);
System.out.println("Please enter a letter:");
char C = input.next().charAt(0);
char [] myArr = new char[N];
int currentStep = 0;
while(currentStep < myArr.length) { // or currentStep < N
    myArr[currentStep] = C; //initialize to C
    currentStep += 1;
}
```
2. Sentinel loops

A sentinel loop keeps track of a boolean variable, and keeps going until that boolean variable becomes false. Here is the algorithm:

```java
boolean sentinel = true;
while(sentinel) {
    // do some stuff
    if(<test>) {
        sentinel = false;
    }
}
```

(or, an alternate version)

```java
while(true) {
    // do some stuff
    if(<test>) {
        break;
    }
}
```
a. Read Strings from the keyboard and print their length, until the user enters the word “STOP!” (all caps).

First, create a Scanner, and copy the loop template:
```java
Scanner input = new Scanner(System.in);
boolean sentinel = true;
while(sentinel) {
    // do some stuff
    if(<test>) {
        sentinel = false;
    }
}
```

Next, add a line to read in a String each time through the loop:
```java
Scanner input = new Scanner(System.in);
boolean sentinel = true;
while(sentinel) {
    String s = input.next();
    if(<test>) {
        sentinel = false;
    }
}
```

Add a line to print the length of the string:
```java
Scanner input = new Scanner(System.in);
boolean sentinel = true;
while(sentinel) {
    String s = input.next();
    System.out.println(s.length());
    if(<test>) {
        sentinel = false;
    }
}
```

Finally, set the test so that it checks for the word “STOP!”:
```java
Scanner input = new Scanner(System.in);
boolean sentinel = true;
while(sentinel) {
    String s = input.next();
    System.out.println(s.length());
    if("STOP!".equals(s)) {
        sentinel = false;
    }
}
```
b. Generate a random number between 1 and 10. Ask the user to guess the number, and keep going until they get it right.

First, create a random number generator, generate a number, and copy the loop template:
```
Random rand = new Random();
int N = rand.nextInt(10) + 1;
boolean sentinel = true;
while(sentinel) {
    // do some stuff
    if(<test>) {
        sentinel = false;
    }
}
```

Next, add statements to the loop that ask the user to guess the number, and read the number from the keyboard.
```
Random rand = new Random();
Scanner input = new Scanner(System.in);
int N = rand.nextInt(10) + 1;
boolean sentinel = true;
while(sentinel) {
    System.out.println("Guess what number I’m thinking (1-10):”);
    int guess = input.nextInt();
    if(<test>) {
        sentinel = false;
    }
}
```

Finally, set the <test> so that the loop stops when the guess matches the random number.
```
Random rand = new Random();
Scanner input = new Scanner(System.in);
int N = rand.nextInt(10) + 1;
boolean sentinel = true;
while(sentinel) {
    System.out.println("Guess what number I’m thinking (1-10):”);
    int guess = input.nextInt();
    if(guess==N) {
        sentinel = false;
    }
}
System.out.println(“Took you long enough.”);
```
3. Accumulation loops

Accumulation loops keep track of and update information each time through the loop. Usually, if a problem says find the “total” or “average” or “maximum” or “biggest” or “anything-est”, then you need an accumulation loop.

Accumulation loops are add-ons to either step-by-step or sentinel loops. You need to start with either a step-by-step or sentinel loop, and then create an extra variable to accumulate extra information.

Here is an algorithm for a basic accumulation step-by-step loop.

```java
int currentStep = START_VALUE;
int (or double/String/it depends) accumulator = 0; (or 0.0,"")
while(currentStep < END_VALUE) {
    int current_value = // something, it depends
    accumulator = ACCUMULATE(accumulator, current_value);
    currentStep += UPDATE_VALUE;
}
```

The data type of the accumulator depends on what kind of information you’re trying to accumulate. Same for the current_value, and the way in which the current_value combines with the existing accumulator value to create the new accumulator value.

For instance, if you’re trying to find the total of an int array, then accumulator is type int, current_value = array[currentStep], and accumulator = accumulator + current_value;

If you’re trying to find the longest String in a String array, then accumulator is type String, current_value = array[currentStep], and accumulator = current_value.length() if current_value.length() is longer than accumulator.length().
a. Read in 10 numbers from the keyboard and compute their average (the accumulator should be the total, then divide by 10 after the loop to get the average).

First, create a Scanner, and copy the loop template:
```java
Scanner input = new Scanner(System.in);
int currentStep = START_VALUE;
int (or double/String/it depends) accumulator = 0; (or 0.0,"")
while(currentStep < END_VALUE) {
    int current_value = // something, it depends
    accumulator = ACCUMULATE(acumulator, current_value);
    currentStep += UPDATE_VALUE;
}
```

Next, set the START_VALUE, END_VALUE, and UPDATE_VALUE so that the loop executes 10 times. It’s easiest to set START_VALUE = 0, END_VALUE = 10, and UPDATE_VALUE = 1.
```java
Scanner input = new Scanner(System.in);
int currentStep = 0;
int (or double/String/it depends) accumulator = 0; (or 0.0,"")
while(currentStep < 10) {
    int current_value = // something, it depends
    accumulator = ACCUMULATE(accumulator, current_value);
    currentStep += 1;
}
```

Next, decide on a data type and initial value for the accumulator and current_value variables. In this case, the program needs to read in either ints or doubles, so either of those will work.
```java
Scanner input = new Scanner(System.in);
int currentStep = 0;
int accumulator = 0;
while(currentStep < 10) {
    int current_value = // something, it depends
    accumulator = ACCUMULATE(accumulator, current_value);
    currentStep += 1;
}
```

Next, write a line that sets the current_value each time through the loop. In this case, the thing that gets repeated is that the program asks the user for a number each time. So the current_value should be the number that the user enters.
```java
Scanner input = new Scanner(System.in);
int currentStep = 0;
int accumulator = 0;
while(currentStep < 10) {
    System.out.println("Please enter a number:");
    int current_value = input.nextInt();
    accumulator = ACCUMULATE(accumulator, current_value);
    currentStep += 1;
```
Then, decide how the accumulator variable should be updated each time through. In this program, the accumulator variable will just add the current_value to the value that used to be in the accumulator.

```java
Scanner input = new Scanner(System.in);
int currentStep = 0;
int accumulator = 0;
while(currentStep < 10) {
    System.out.println("Please enter a number:");
    int current_value = input.nextInt();
    accumulator = accumulator + current_value;
    currentStep += 1;
}
```

Finally, do something with the accumulator after it’s done accumulating stuff. In this case, divide it by 10 and print the average. Note that in order to compute an exact average, I have to divide by 10.0 to ensure that Java uses double arithmetic instead of integer arithmetic.

```java
Scanner input = new Scanner(System.in);
int currentStep = 0;
int accumulator = 0;
while(currentStep < 10) {
    System.out.println("Please enter a number:");
    int current_value = input.nextInt();
    accumulator = accumulator + current_value;
    currentStep += 1;
}

double average = accumulator / 10.0;
System.out.println("average = "+average);
```
b. Read in 10 Strings from the keyboard and compute the total length of all of them.

First, create a Scanner, and copy the loop template:
```java
Scanner input = new Scanner(System.in);
int currentStep = START_VALUE;
int (or double/String/it depends) accumulator = 0; (or 0.0,"")
while(currentStep < END_VALUE) {
    int current_value = // something, it depends
    accumulator = ACCUMULATE(accumulator, current_value);
    currentStep += UPDATE_VALUE;
}
```

Second, set the START/END/UPDATE_VALUEs to make this go 10 times.
```java
Scanner input = new Scanner(System.in);
int currentStep = 0;
int (or double/String/it depends) accumulator = 0; (or 0.0,"")
while(currentStep < 10) {
    int current_value = // something, it depends
    accumulator = ACCUMULATE(accumulator, current_value);
    currentStep += 1;
}
```

Third, decide on a data type and initial value for the accumulator, and a data type for the current_value. In this case, the accumulator is storing the total length of the Strings seen so far. Since length is an integer-type, we want the accumulator to be an int. Since the accumulator hasn’t seen any strings to start with, we want its initial value to be zero.

On the other hand, the current_value variable is a String, since we’re reading in Strings. So the data type for current_value does not match the data type of the accumulator in this example.
```java
Scanner input = new Scanner(System.in);
int currentStep = 0;
int accumulator = 0;
while(currentStep < 10) {
    String current_value = // something, it depends
    accumulator = ACCUMULATE(accumulator, current_value);
    currentStep += 1;
}
```

Next, read in a String each time through the loop, and set the current_value to that.
```java
Scanner input = new Scanner(System.in);
int currentStep = 0;
int accumulator = 0;
while(currentStep < 10) {
    String current_value = input.next();
    accumulator = ACCUMULATE(accumulator, current_value);
    currentStep += 1;
}
Then, determine how to update the accumulator. In this example, we want the accumulator to take into account the sum of the lengths of the Strings it has already seen (the current value of accumulator) and the length of the current string (current_value.length()) to compute the length of all the Strings it has seen so far. I added a line to print out the total length at the end.

```java
Scanner input = new Scanner(System.in);
int currentStep = 0;
int accumulator = 0;
while(currentStep < 10) {
    String current_value = input.next();
    accumulator = accumulator + current_value.length();
    currentStep += 1;
}
System.out.println("the length of all strings is " + accumulator);
```
c. Read in Strings from the keyboard until the user types “STOP!”.
Compute the total length of all of them. (Note: this is an accumulation loop with a sentinel loop, not an accumulation loop with a step-by-step loop.)

First, create a Scanner, and copy the loop template for a sentinel loop:

```java
Scanner input = new Scanner(System.in);
boolean sentinel = true;
while(sentinel) {
    // do some stuff
    if(<test>) {
        sentinel = false;
    }
}
```

Second, add in template statements for an accumulation loop:

```java
Scanner input = new Scanner(System.in);
boolean sentinel = true;
int/double/String/something accumulator = INITIAL_VALUE;
while(sentinel) {
    int/double/String/something current_value = // new value
    accumulator = ACCUMULATE(accumulator, current_value);
    if(<test>) {
        sentinel = false;
    }
}
```

Third, decide on a data type and initial value for the accumulator. Since we’re doing length again, data type int and INITIAL_VALUE = 0.

```java
Scanner input = new Scanner(System.in);
boolean sentinel = true;
int accumulator = 0;
while(sentinel) {
    int/double/String/something current_value = // new value
    accumulator = ACCUMULATE(accumulator, current_value);
    if(<test>) {
        sentinel = false;
    }
}
```

Then, decide on a data type and statement to set the current_value each time through the loop. In this problem, we need to read a String each time, so that will be the current_value.

```java
Scanner input = new Scanner(System.in);
boolean sentinel = true;
int accumulator = 0;
while(sentinel) {
    String current_value = input.next();
    accumulator = ACCUMULATE(accumulator, current_value);
```
if(<test>) {
    sentinel = false;
}

Then, decide on how to update the accumulator. In this problem, it’s just like the previous problem.
Scanner input = new Scanner(System.in);
boolean sentinel = true;
int accumulator = 0;
while(sentinel) {
    String current_value = input.next();
    accumulator = accumulator + current_value.length();
    if(<test>) {
        sentinel = false;
    }
}

Finally, set the <test> so that it checks for the word “STOP!”.
Scanner input = new Scanner(System.in);
boolean sentinel = true;
int accumulator = 0;
while(sentinel) {
    String current_value = input.next();
    accumulator = accumulator + current_value.length();
    if(current_value.equals(“STOP!”)) {
        sentinel = false;
    }
}
System.out.println(“total length = “ + accumulator);
d. Read in 10 numbers from the keyboard and count how many are odd (or also try counting how many are divisible by 7, or how many are not divisible by either 12 or 19).

First, create a Scanner, and copy the loop template.
Scanner input = new Scanner(System.in);
int currentStep = START VALUE;
int (or double/String/it depends) accumulator = 0; (or 0.0,"")
while(currentStep < END VALUE) {
    int current_value = // something, it depends
    accumulator = ACCUMULATE(accumulator, current_value);
    currentStep += UPDATE_VALUE;
}

Second, decide on START/END/UPDATE_VALUEs to make the loop happen 10 times.
Scanner input = new Scanner(System.in);
int currentStep = 0;
int (or double/String/it depends) accumulator = 0; (or 0.0,"")
while(currentStep < 10) {
    int current_value = // something, it depends
    accumulator = ACCUMULATE(accumulator, current_value);
    currentStep += 1;
}

Third, decide on a data type and initial value for the accumulator. Since the accumulator represents the number of times we’ve seen something happen, we should make it an int and start it at 0.
Scanner input = new Scanner(System.in);
int currentStep = 0;
int accumulator = 0;
while(currentStep < 10) {
    int current_value = // something, it depends
    accumulator = ACCUMULATE(accumulator, current_value);
    currentStep += 1;
}

Fourth, determine a data type and statement to set the current_value each time through the loop.
Scanner input = new Scanner(System.in);
int currentStep = 0;
int accumulator = 0;
while(currentStep < 10) {
    int current_value = input.nextInt();
    accumulator = ACCUMULATE(accumulator, current_value);
    currentStep += 1;
}

Finally, determine an update statement for the accumulator. In this case, we want the accumulator to represent the number of times we have seen an odd number (or number divisible
by 7, or a more complicated test). When the accumulator counts something, that almost always means you need an if() statement for the update of the accumulator. The test of the if() statement will check to see if the current_value is odd (or divisible by 7, or something more complicated), and then the actual update will just increase the accumulator by 1 if the test is true.

Scanner input = new Scanner(System.in);
int currentStep = 0;
int accumulator = 0;
while(currentStep < 10) {
    int current_value = input.nextInt();
    // test to see if current_value is odd
    if(current_value % 2 == 0) {
        accumulator = accumulator + 1;
    }
    currentStep += 1;
}
System.out.println("number of odd values: " + accumulator);

To change this program to something that counts the number of values that are divisible by 7, the only thing you need to change is the test in the if() statement:

    if(current_value % 7 == 0) {

To change this program to something that counts the number of values that are NOT divisible by either 12 or 19, again the only thing you need to change is the test in the if() statement:

    if(current_value % 12 != 0 && current_value % 19 != 0) {
e. Read in x and y coordinates of Point objects. Find the Point that is farthest from the origin (or new Point(0,0)). Stop when the user enters 0 for x and 0 for y.

This is another accumulator with a sentinel loop. Start by creating a Scanner and copying the template for the accumulator+sentinel loop from above:

```java
Scanner input = new Scanner(System.in);
boolean sentinel = true;
int/double/String/something accumulator = INITIAL_VALUE;
while(sentinel) {
    int/double/String/something current_value = // new value
    accumulator = ACCUMULATE(accumulator, current_value);
    if(<test>) {
        sentinel = false;
    }
}
```

Next, decide on a data type and initial value for the accumulator. In this case, we want the accumulator to represent the Point that is farthest from the origin. So we want its data type to be Point, and we want to start it at the origin:

```java
Scanner input = new Scanner(System.in);
boolean sentinel = true;
Point accumulator = new Point(0,0); // or just "= new Point();"
while(sentinel) {
    int/double/String/something current_value = // new value
    accumulator = ACCUMULATE(accumulator, current_value);
    if(<test>) {
        sentinel = false;
    }
}
```

Next, set the current_value. In this example, the current_value is a Point. Reading in a point means reading in both an x and a y coordinate, so that takes a few more lines than usual:

```java
Scanner input = new Scanner(System.in);
boolean sentinel = true;
Point accumulator = new Point(0,0); //
while(sentinel) {
    int x = input.nextInt();
    int y = input.nextInt();
    Point current_value = new Point(x, y);
    accumulator = ACCUMULATE(accumulator, current_value);
    if(<test>) {
        sentinel = false;
    }
}
```
Next, update the accumulator. Any time you have to find the something-est value (big-est, small-est, farth-est from the origin, long-est, tall-est, whatever), you will need an if() statement for the accumulator. The if() statement will test whether the current value is big-er (or small-er, farth-er from the origin, long-er, or tall-er) than the accumulator. If the test is true, then inside the if statement the accumulator will be set equal to the current_value:

```java
Scanner input = new Scanner(System.in);
boolean sentinel = true;
Point accumulator = new Point(0,0);
Point origin = new Point();
while(sentinel) {
    int x = input.nextInt();
    int y = input.nextInt();
    Point current_value = new Point(x, y);
    if(origin.distance(current_value) > origin.distance(accumulator)) {
        accumulator = current_value;
    }
    if(<test>) {
        sentinel = false;
    }
}
```

Finally, set the test to check to see if the user entered a value to make the loop stop:

```java
Scanner input = new Scanner(System.in);
boolean sentinel = true;
Point accumulator = new Point(0,0);
Point origin = new Point();
while(sentinel) {
    int x = input.nextInt();
    int y = input.nextInt();
    Point current_value = new Point(x, y);
    if(origin.distance(current_value) > origin.distance(accumulator)) {
        accumulator = current_value;
    }
    if(origin.equals(current_value)) {
        sentinel = false;
    }
}
```
f. Create an array of 100 doubles, and initialize the array with random Gaussian values (use the `nextGaussian()` method from the Random class, which returns a double). Compute the mean (average) and standard deviation of the array. If the mean is \( M \), then you can compute the standard deviation by computing \( S = \sum_i (x[i]-M)^2 \), and then taking the square root of \( S \).

This problem involves 2 loops. The first initializes the values in the array and accumulates the sum (so we can get the mean), and the second does accumulates the standard deviation. Since the standard deviation computation uses the value of the mean, you can’t start computing it until you’ve finished computing the mean, so you need a second loop for it.

First, create a random-number generator, an array of 100 doubles, and copy the loop template.
```java
Random rand = new Random();
double [] dArr = new double[100];
int currentStep = START_VALUE;
int (or double/String/it depends) accumulator = 0; (or 0.0,"")
while(currentStep < END_VALUE) {
    int current_value = // something, it depends
    accumulator = ACCUMULATE(accumulator, current_value);
    currentStep += UPDATE_VALUE;
}
```

Second, set the START/END/UPDATE values.
```java
Random rand = new Random();
double [] dArr = new double[100];
int currentStep = 0;
int (or double/String/it depends) accumulator = 0; (or 0.0,"")
while(currentStep < dArr.length) {
    int current_value = // something, it depends
    accumulator = ACCUMULATE(accumulator, current_value);
    currentStep += 1;
}
```

Third, set the accumulator data type and initial value. For the first loop, we want to accumulate the sum. Since we’re adding doubles, we’ll make the accumulator a double, and we’ll start it at 0.
```java
Random rand = new Random();
double [] dArr = new double[100];
int currentStep = 0;
double accumulator = 0.0;
while(currentStep < dArr.length) {
    int current_value = // something, it depends
    accumulator = ACCUMULATE(accumulator, current_value);
    currentStep += 1;
}
```
Fourth, set the current value using the Random class’s nextGaussian() method.

```java
Random rand = new Random();
double[] dArr = new double[100];
int currentStep = 0;
double accumulator = 0.0;
while(currentStep < dArr.length) {
    int current_value = rand.nextGaussian();
    accumulator = ACCUMULATE(accumulator, current_value);
    currentStep += 1;
}
```

Fifth, update the accumulator. We’re just doing a sum, so this one’s a relatively easy one.

```java
Random rand = new Random();
double[] dArr = new double[100];
int currentStep = 0;
double accumulator = 0.0;
while(currentStep < dArr.length) {
    int current_value = rand.nextGaussian();
    accumulator = accumulator + current_value;
    currentStep += 1;
}
```

Compute the mean.

```java
Random rand = new Random();
double[] dArr = new double[100];
int currentStep = 0;
double accumulator = 0.0;
while(currentStep < dArr.length) {
    int current_value = rand.nextGaussian();
    accumulator = accumulator + current_value;
    currentStep += 1;
}
// double arithmetic b/c accumulator is type double
double mean = accumulator / dArr.length;
```

Ok, that’s the end of the first loop. But now we need to compute the standard deviation, so copy another accumulator loop template below this one.

```java
Random rand = new Random();
double[] dArr = new double[100];
int currentStep = 0;
double accumulator = 0.0;
while(currentStep < dArr.length) {
    int current_value = rand.nextGaussian();
    accumulator = accumulator + current_value;
    currentStep += 1;
}
```
double mean = accumulator / dArr.length;
// be careful to use different variable names for accumulator
// and currentStep this time through
int currentStep2 = START_VALUE;
int (or double/String/it depends) accumulator2 = 0; (or 0.0,"")
while(currentStep2 < END_VALUE) {
    int current_value2 //= something, it depends
    accumulator2 = ACCUMULATE(accumulator2, current_value2);
    currentStep2 += UPDATE_VALUE;
}

Set the START/END/UPDATE values for the new accumulator loop.
Random rand = new Random();
double [] dArr = new double[100];
int currentStep = 0;
double accumulator = 0.0;
while(currentStep < dArr.length) {
    int current_value = rand.nextGaussian();
    accumulator = accumulator + current_value;
    currentStep += 1;
}
double mean = accumulator / dArr.length;
int currentStep2 = 0;
int (or double/String/it depends) accumulator2 = 0; (or 0.0,"")
while(currentStep2 < dArr.length) {
    int current_value2 //= something, it depends
    accumulator2 = ACCUMULATE(accumulator2, current_value2);
    currentStep2 += 1;
}

Set the accumulator2 to type double and initialize to zero, since we’re accumulating the sum of doubles.
Random rand = new Random();
double [] dArr = new double[100];
int currentStep = 0;
double accumulator = 0.0;
while(currentStep < dArr.length) {
    int current_value = rand.nextGaussian();
    accumulator = accumulator + current_value;
    currentStep += 1;
}
double mean = accumulator / dArr.length;
int currentStep2 = 0;
double accumulator2 = 0;
while(currentStep2 < dArr.length) {
    int current_value2 //= something, it depends
    accumulator2 = ACCUMULATE(accumulator2, current_value2);
}
currentStep2 += 1;
}

Set current_value2 to be the contribution from dArr[currentStep2] to the standard deviation. Use the formula above.
Random rand = new Random();
double [] dArr = new double[100];
int currentStep = 0;
double accumulator = 0.0;
while(currentStep < dArr.length) {
    int current_value = rand.nextGaussian();
    accumulator = accumulator + current_value;
    currentStep += 1;
}

double mean = accumulator / dArr.length;
int currentStep2 = 0;
double accumulator2 = 0;
while(currentStep2 < dArr.length) {
    int current_value2 =
        (dArr[currentStep2]-mean)* (dArr[currentStep2]-mean);
    accumulator2 = ACCUMULATE(accumulator2, current_value2);
    currentStep2 += 1;
}

Update the accumulator2. It’s just a sum, so again it’s relatively easy.
Random rand = new Random();
double [] dArr = new double[100];
int currentStep = 0;
double accumulator = 0.0;
while(currentStep < dArr.length) {
    int current_value = rand.nextGaussian();
    accumulator = accumulator + current_value;
    currentStep += 1;
}

double mean = accumulator / dArr.length;
int currentStep2 = 0;
double accumulator2 = 0;
while(currentStep2 < dArr.length) {
    int current_value2 =
        (dArr[currentStep2]-mean)* (dArr[currentStep2]-mean);
    accumulator2 = accumulator2 + current_value2;
    currentStep2 += 1;
}
Finally, take the square root of accumulator2 to get the standard deviation.

```java
Random rand = new Random();
double[] dArr = new double[100];
int currentStep = 0;
double accumulator = 0.0;
while(currentStep < dArr.length) {
    int current_value = rand.nextGaussian();
    accumulator = accumulator + current_value;
    currentStep += 1;
}
double mean = accumulator / dArr.length;
int currentStep2 = 0;
double accumulator2 = 0;
while(currentStep2 < dArr.length) {
    int current_value2 =
        (dArr[currentStep2]-mean) * (dArr[currentStep2]-mean);
    accumulator2 = accumulator2 + current_value2;
    currentStep2 += 1;
}
double stdDev = Math.sqrt(accumulator2);

System.out.println("mean = " + mean + ", std dev = " + stdDev);
```
g. Create an array of 100 Strings, and initialize the array by generating a random integer between 0 and 1 million for each element, and converting the integer to a String. Find the longest String in the array.

Conceptually, there are two parts to this: initializing the array, and finding the longest String. We’ll do them as two separate loops (although you could do them all at once). First, construct the array, and copy the step-by-step loop template for the initialization loop.

Random rand = new Random();
String [] myArr = new String[100];
int currentStep = START_VALUE;
while(currentStep < END_VALUE) {
    // do something
    currentStep += UPDATE_VALUE;
}

Second, set the START/END/UPDATE values.

Random rand = new Random();
String [] myArr = new String[100];
int currentStep = 0;
while(currentStep < myArr.length) {
    // do something
    currentStep += 1;
}

Third, create a random number between 0 and 1 million, convert it to a String, and store it at the currentStep index in the array. To convert the int to a String, I just concatenate it with the empty String "".

Random rand = new Random();
String [] myArr = new String[100];
int currentStep = 0;
while(currentStep < myArr.length) {
    myArr[currentStep] = "" + rand.nextInt(1000001);
    currentStep += 1;
}

Ok, that’s it for initialization, now it’s time for accumulation. Copy the template for the accumulation loop below the initialization loop. Make sure to use a different name for the “currentStep” variable in the second template, to avoid a conflict with the first one.

Random rand = new Random();
String [] myArr = new String[100];
int currentStep = 0;
while(currentStep < myArr.length) {
    myArr[currentStep] = "" + rand.nextInt(1000001);
    currentStep += 1;
}
int currentStep2 = START_VALUE;
int (or double/String/it depends) accumulator = 0; (or 0.0,""
while(currentStep2 < END_VALUE) {
    int current_value = // something, it depends
    accumulator = ACCUMULATE(accumulator, current_value);
    currentStep2 += UPDATE_VALUE;
}

Set the START/END/UPDATE values.
Random rand = new Random();
String [] myArr = new String[100];
int currentStep = 0;
while(currentStep < myArr.length) {
    myArr[currentStep] = "" + rand.nextInt(1000001);
    currentStep += 1;
}
int currentStep2 = 0;
int (or double/String/it depends) accumulator = 0; (or 0.0,""")
while(currentStep2 < myArr.length) {
    int current_value = // something, it depends
    accumulator = ACCUMULATE(accumulator, current_value);
    currentStep2 += 1;
}

Set the data type and initial value for the accumulator. We want the longest String in the array, so we want the accumulator to be a String. We’ll initialize it to the empty String, which has length 0 and is the shortest possible String.
Random rand = new Random();
String [] myArr = new String[100];
int currentStep = 0;
while(currentStep < myArr.length) {
    myArr[currentStep] = "" + rand.nextInt(1000001);
    currentStep += 1;
}
int currentStep2 = 0;
String accumulator = "";
while(currentStep2 < myArr.length) {
    int current_value = // something, it depends
    accumulator = ACCUMULATE(accumulator, current_value);
    currentStep2 += 1;
}

Set the current_value to be the value at the currentStep index of the array.
Random rand = new Random();
String [] myArr = new String[100];
int currentStep = 0;
while(currentStep < myArr.length) {
    myArr[currentStep] = "" + rand.nextInt(1000001);
    currentStep += 1;
As mentioned before, any time you need to find the X-est of something (smallest, longest, biggest, tallest, etc.), you need an if() statement for the accumulation step. The test will check to see if the accumulator is X-er than the current_value. If the current_value is X-er (in this case, longer), then the body of the if statement will set the accumulator equal to the current_value.

```java
int currentStep2 = 0;
String accumulator = "";
while(currentStep2 < myArr.length) {
    String current_value = myArr[currentStep];
    accumulator = ACCUMULATE(accumulator, current_value);
    currentStep2 += 1;
}
```
Nested Loops

Nested loops are the toughest ones we have seen. They combine an inner loop (which repeats some stuff X times) with an outer loop (which repeats the whole inner loop Y times). As a result, the inner loop’s stuff gets repeated $X \times Y$ times.

A nested loop can combine an inner step-by-step loop with an outer sentinel loop, or an inner step-by-step loop with an outer step-by-step loop, or any other combination. It depends on the problem.

Some common versions:
- Displaying two-dimensional figures (ASCII art) requires inner and outer step-by-step loops.
- Computing with 2D arrays normally requires inner and outer step-by-step loops, where the inner loop repeats as many times as the number of columns and the outer loop repeats as many times as the number of rows.
- File processing often involves nested sentinel loops (we’ll get to those later).

It’s also possible to combine accumulation loops with nested loops.
a. Create a 10 x 10 2D array of Point objects, initialized randomly.

First, create the array and a random number generator.

```java
Random rand = new Random();
Point [][] myArr = new Point[10][10];
int row = START_ROW;
while(row < END_ROW) {
    int col = START_COL;
    while(col < END_COL) {
        // do stuff
        col = col + UPDATE_COL;
    }
    row = row + UPDATE_ROW;
}
```

Then, set the START/END/UPDATE for the rows and columns. The row should go from 0 to 1 less than myArr.length. The col should go from 0 to 1 less than myArr[row].length.

```java
Random rand = new Random();
Point [][] myArr = new Point[10][10];
int row = 0;
while(row < myArr.length) {
    int col = 0;
    while(col < myArr[row].length) {
        // do stuff
        col = col + 1;
    }
    row = row + 1;
}
```

Finally, generate a random x and y coordinate, and use those to construct a Point object. Store that point object in the array at the current row and col. The problem doesn’t specify what kind of random number, so I’ll generate x coordinates between 1 and 10, and y coordinates between 7 and 22.

```java
Random rand = new Random();
Point [][] myArr = new Point[10][10];
int row = 0;
while(row < myArr.length) {
    int col = 0;
    while(col < myArr[row].length) {
        int x = rand.nextInt(10)+1;
        int y = rand.nextInt(16)+7; //range 16 means [0–15]
        myArr[row][col] = new Point(x, y);
        col = col + 1;
    }
    row = row + 1;
}
```
b. Create a 8 x 7 2D array, and initialize it randomly (use either the nextGaussian() or nextInt() method from the Random class, you choose). Then compute the sum, average, largest, and smallest elements in the array. (Initialization is a nested step-by-step loop, the rest is a nested step-by-step loop plus accumulation).

First, create the array and a random number generator.
Random rand = new Random();
double [][] myArr = new double[8][7];
int row = START_ROW;
while(row < END_ROW) {
    int col = START_COL;
    while(col < END_COL) {
        // do stuff
        col = col + UPDATE_COL;
    }
    row = row + UPDATE_ROW;
}

Second, set the START/END/UPDATE values.
Random rand = new Random();
double [][] myArr = new double[8][7];
int row = 0;
while(row < myArr.length) {
    int col = 0;
    while(col < myArr[row].length) {
        // do stuff
        col = col + 1;
    }
    row = row + 1;
}

Third, initialize each element using the rand.nextGaussian() method.
Random rand = new Random();
double [][] myArr = new double[8][7];
int row = 0;
while(row < myArr.length) {
    int col = 0;
    while(col < myArr[row].length) {
        myArr[row][col] = rand.nextGaussian();
        col = col + 1;
    }
    row = row + 1;
}
Fourth, turn this into an accumulation loop. The problem asks for 4 kinds of accumulators: sum, average, max, and min. Average can be computed from the sum, so we'll create three variables as accumulators: accumSum, accumMax, and accumMin. We'll initialize accumMax to the smallest possible double value (Double.MIN_VALUE), so that any value in the array will be bigger than it. We'll initialize accumMin to the largest possible double value (Double.MAX_VALUE) so that any value in the array will be smaller than it. We'll also create a current_value variable.

```
Random rand = new Random();
double accumSum = 0;
double accumMax = Double.MIN_VALUE;
double accumMin = Double.MAX_VALUE;
double [][] myArr = new double[8][7];
int row = 0;
while(row < myArr.length) {
    int col = 0;
    while(col < myArr[row].length) {
        myArr[row][col] = rand.nextGaussian();
        double current_value = myArr[row][col];
        col = col + 1;
    }
    row = row + 1;
}
```

Finally, add accumulation statements. You've seen similar ones to each of these before.

```
Random rand = new Random();
double accumSum = 0;
double accumMax = Double.MIN_VALUE;
double accumMin = Double.MAX_VALUE;
double [][] myArr = new double[8][7];
int row = 0;
while(row < myArr.length) {
    int col = 0;
    while(col < myArr[row].length) {
        myArr[row][col] = rand.nextGaussian();
        double current_value = myArr[row][col];
        accumSum = accumSum + current_value;
        if(current_value > accumMax) {
            accumMax = current_value;
        }
        if(current_value < accumMin) {
            accumMin = current_value;
        }
        col = col + 1;
    }
    row = row + 1;
}
```
c. Create an array of 10 Strings, each initialized with random integers between 0 and 1 million (converted to Strings). Count how many 0’s appear in all of the Strings. (Initialization is a single step-by-step loop. Counting is a nested step-by-step loop with accumulation.)

First, create the array and a random number generator, and copy the step-by-step loop template (for initialization).

```java
Random rand = new Random();
String [] myArr = new String[10];
int currentStep = START_VALUE;
while(currentStep < END_VALUE) {
    // do something, which may involve the value of currentStep
    currentStep += UPDATE_VALUE;
}
```

Set the START/END/UPDATE values.

```java
Random rand = new Random();
String [] myArr = new String[10];
int currentStep = 0;
while(currentStep < myArr.length) {
    // do something, which may involve the value of currentStep
    currentStep += 1;
}
```

Create a random number, convert it to a String, and assign it to the array at index currentStep.

```java
Random rand = new Random();
String [] myArr = new String[10];
int currentStep = 0;
while(currentStep < myArr.length) {
    myArr[currentStep] = "" + rand.nextInt(1000001);
    currentStep += 1;
}
```

That’s it for initialization. Now we need to add a nested accumulation loop to count the number of zeros in the Strings. Copy the template, and make it nested:

```java
Random rand = new Random();
String [] myArr = new String[10];
int currentStep = 0;
while(currentStep < myArr.length) {
    myArr[currentStep] = "" + rand.nextInt(1000001);
    currentStep += 1;
}
```

```java
int currentStep2 = START_VALUE;
int (or double/String/it depends) accumulator = 0; (or 0.0,"")
while(currentStep2 < END_VALUE) {
    int currentStep3 = START_VALUE;
```
while(currentStep3 < END_VALUE) {
    int current_value = // something, it depends
    accumulator = ACCUMULATE(accumulator, current_value);
    currentStep3 += UPDATE_VALUE;
}
currentStep2 += UPDATE_VALUE;
}

Set the START/END/UPDATE values. currentStep2 goes through each array index (0 up to myArr.length-1). currentStep3 goes through each String position (0 up to myArr[currentStep2].length()-1).
Random rand = new Random();
String [] myArr = new String[10];
int currentStep = 0;
while(currentStep < myArr.length) {
    myArr[currentStep] = "" + rand.nextInt(1000001);
    currentStep += 1;
}
int currentStep2 = 0;
int (or double/String/it depends) accumulator = 0; (or 0.0,""")
while(currentStep2 < myArr.length) {
    int currentStep3 = 0;
    while(currentStep3 < myArr[currentStep2].length()) {
        int current_value = // something, it depends
        accumulator = ACCUMULATE(accumulator, current_value);
        currentStep3 += 1;
    }
    currentStep2 += 1;
}

Set the accumulator type to int (since we’re counting things), and start it at 0.
Random rand = new Random();
String [] myArr = new String[10];
int currentStep = 0;
while(currentStep < myArr.length) {
    myArr[currentStep] = "" + rand.nextInt(1000001);
    currentStep += 1;
}
int currentStep2 = 0;
int accumulator = 0;
while(currentStep2 < myArr.length) {
    int currentStep3 = 0;
    while(currentStep3 < myArr[currentStep2].length()) {
        int current_value = // something, it depends
accumulator = ACCUMULATE(accumulator, current_value);
currentStep3 += 1;
}
currentStep2 += 1;
}

Set the current value to be the current letter of the current String. That’s myArr[currentStep2].charAt(currentStep3).
Random rand = new Random();
String [] myArr = new String[10];
int currentStep = 0;
while(currentStep < myArr.length) {
    myArr[currentStep] = "" + rand.nextInt(1000001);
    currentStep += 1;
}

int currentStep2 = 0;
int accumulator = 0;
while(currentStep2 < myArr.length) {
    int currentStep3 = 0;
    while(currentStep3 < myArr[currentStep2].length()) {
        char current_value =
        myArr[currentStep2].charAt(currentStep3);
        accumulator = ACCUMULATE(accumulator, current_value);
        currentStep3 += 1;
    }
    currentStep2 += 1;
}

Finally, do the accumulation step. Since we’re counting, we need an if() statement that tests whether the thing we’re counting actually happens at the current step. If it does (test is true), then the body of the if statement increases the accumulator by 1. In this case, the test is to see if the current letter (stored in variable current_value) is a zero character, or ‘0’.
Random rand = new Random();
String [] myArr = new String[10];
int currentStep = 0;
while(currentStep < myArr.length) {
    myArr[currentStep] = "" + rand.nextInt(1000001);
    currentStep += 1;
}

int currentStep2 = 0;
int accumulator = 0;
while(currentStep2 < myArr.length) {
    int currentStep3 = 0;
    while(currentStep3 < myArr[currentStep2].length()) {
        char current_value =
    }
myArr[currentStep2].charAt(currentStep3);
    if(current_value=='0') {
        accumulator++;
    }
    currentStep3 += 1;
}
currentStep2 += 1;

System.out.println("number of 0's = " + accumulator);
d. Create a 4 x 2 array A, a 2 x 3 array B, and a 4 x 3 array C. Initialize A and B to random values (either nextGaussian() or nextInt(), you can choose). Then store the result of A * B in C, where A* B is matrix multiplication. That means, for element C[row][column] you add up \( \sum_j A[row][j] * B[j][col] \). (Double-nested step-by-step loops with accumulation)

This is a tough one – it involves a double-nested loop with accumulation. Start by creating and initializing the arrays A, B, and C.

```java
Random rand = new Random();
double [][] A = new double[4][2];
double [][] B = new double[2][3];
double [][] C = new double[4][3];

//initialize A and B to random Gaussian-distributed numbers
//leave C at all 0.0 for now
for(int row = 0; row<A.length; row++) {
    for(int col = 0; col<A[row].length; col++) {
        A[row][col] = rand.nextGaussian();
    }
}

for(int row = 0; row<B.length; row++) {
    for(int col = 0; col<B[row].length; col++) {
        B[row][col] = rand.nextGaussian();
    }
}

C[0][0] = 0.0; // use C[0][0] as accumulator, initialize to 0.0
```

Next step – do an accumulation loop for cell [0][0] of array C.

```java
Random rand = new Random();
double [][] A = new double[4][2];
double [][] B = new double[2][3];
double [][] C = new double[4][3];

//initialize A and B to random Gaussian-distributed numbers
//leave C at all 0.0 for now
for(int row = 0; row<A.length; row++) {
    for(int col = 0; col<A[row].length; col++) {
        A[row][col] = rand.nextGaussian();
    }
}

for(int row = 0; row<B.length; row++) {
    for(int col = 0; col<B[row].length; col++) {
        B[row][col] = rand.nextGaussian();
    }
}

C[0][0] = 0.0; // use C[0][0] as accumulator, initialize to 0.0
```
for(int j=0; j<A[0].length; j++) {
    double current_value = A[0][j] * B[j][0];
    C[0][0] = C[0][0] + current_value;
}

Now, take that accumulation loop and put an outer loop around it, so that it works for every column in the row C[0] (in other words, it should work for C[0][0], C[0][1], and C[0][2]).

Random rand = new Random();
double [][] A = new double[4][2];
double [][] B = new double[2][3];
double [][] C = new double[4][3];

//initialize A and B to random Gaussian-distributed numbers
//leave C at all 0.0 for now
for(int row = 0; row<A.length; row++) {
    for(int col = 0; col<A[row].length; col++) {
        A[row][col] = rand.nextGaussian();
    }
}
for(int row = 0; row<B.length; row++) {
    for(int col = 0; col<B[row].length; col++) {
        B[row][col] = rand.nextGaussian();
    }
}
for(int col = 0; col<C[0].length; col++) {
    C[0][col] = 0.0;  // generalize C[0][0] to C[0][col]
    for(int j=0; j<A[0].length; j++) {
        double current_value = A[0][j] * B[j][col];
        C[0][col] = C[0][col] + current_value;
    }
}

Finally, take that accumulation loop and put another outer loop around it, so that it works for every column and every row in C.

Random rand = new Random();
double [][] A = new double[4][2];
double [][] B = new double[2][3];
double [][] C = new double[4][3];

//initialize A and B to random Gaussian-distributed numbers
//leave C at all 0.0 for now
for(int row = 0; row<A.length; row++) {
    for(int col = 0; col<A[row].length; col++) {
        A[row][col] = rand.nextGaussian();
    }
}
for(int row = 0; row < B.length; row++) {
    for(int col = 0; col < B[row].length; col++) {
        B[row][col] = rand.nextGaussian();
    }
}

for(int row = 0; row < C.length; row++) {
    for(int col = 0; col < C[row].length; col++) {
        C[row][col] = 0.0;
        for(int j = 0; j < A[row].length; j++) {
            double current_value = A[row][j] * B[j][col];
            C[row][col] = C[row][col] + current_value;
        }
    }
}