Factorials.java

```java
class Factorials {
  static int factorial1(int n) {
    System.out.println("factorial1(" + n + ")");
    int total = 1;
    System.out.println("i\ttotal before\ttotal after");
    for(int i = 1; i <= n; i += 1) {
      System.out.print(i + "\t\t" + total);
      total = total * i;
      System.out.println("\t\t" + total);
    }
    return total;
  }


  static int factorial2(int n) {
    System.out.println("factorial2(" + n + ")");
    int total = 1;
    System.out.println("i\ttotal before\ttotal after");
    for(int i = n; i >= 1; i -= 1) {
      System.out.print(i + "\t\t" + total);
      total = total * i;
      System.out.println("\t\t" + total);
    }
    return total;
  }

  public static void main(String[] args) {
    assert factorial1(5) == 120;
    assert factorial2(5) == 120;
  }

}
```

at terminal

```
$ javac Factorials.java
$ java Factorials
factorial1(5)
i       total before    total after
1               1               1
2               1               2









factorial2(5)
i       total before    total after
5               1               5
4               5               20
```

```
for(<initialize>; <check>; <update>) {
  <body>
}
```

To evaluate a 3-clause for loop:

- Evaluate <initialize> (this only happens once)
- Evaluate <check>
  - If the result is false, end the loop
  - If the result is true:
    - Evaluate <body>
    - Evaluate <update>
    - Go back to "Evaluate <check>"

<check> must evaluate to a boolean
<initialize> typically declares or initializes a variable
<update> typically changes a variable
<body> does some calculation of one step of the answer

```
for(<type> <name>: <collection>) {
  <body>
}
```

To evaluate an iterating for loop (or **enhanced for loop**)

- Evaluate <collection> to a value
- For each element in collection (e.g. each array element) in order
  - Store that element in <name>
  - Evaluate <body>

Examples on other side of sheet.

```
<type>[] <name> = { <e1>, <e2>, ... };
```

Creates an **array** and stores it in the variable <name>.

All elements e1, e2, must have the given type.

Examples on other side of sheet.

```
<array>[<index>]
```

Arrays can be indexed as in Python. <index> should evaluate to an int, and <array> to an array value. Indices start at 0.

Examples on other side of sheet.

```java
class ArrayExamples {

  static int product(int[] nums) {
    int total = 1;
    for(int n: nums) {
      total *= n;
    }
    return total;
  }
  static {
    int[] nums1 = {1, 2, 3, 5};
    assert product(nums1) == 30;
    int[] nums2 = {4, 2, 3};
    assert product(nums2) == 24;
  }

  static int max(int[] nums) {
    int biggest = nums[0];
    for(int n: nums) {
      if(n > biggest) { biggest = n; }
    }
    return biggest;
  }
  static {
    int[] nums1 = {50, 60, 70, 30};
    assert max(nums1) == 70;
    int[] nums2 = {30, 50, 60, 10, 90};
    assert max(nums2) == 90;
    int[] nums3 = {90};
    assert max(nums3) == 90;
  }

  static int find(String[] strs, String tofind) {
    for(int index = 0; index < strs.length; index += 1) {
      if(strs[index].equals(tofind)) { return index; }
    }
    return -1;
  }
  static {
    String[] abc = {"a", "b", "c"};
    assert find(abc, "b") == 1;
    assert find(abc, "d") == -1;
    assert find(abc, "c") == 2;
  }

  static int sumAlternating(int[] nums) {
    int total = 0;
    for(int i = 0; i < nums.length; i += 2) {
      total += nums[i];
    }
    return total;
  }
  static {








  }


  public static void main(String[] args) {
    // All the blocks above run the assertions
  }
}
```

ArrayExamples.java