Looping “gotchas”
- sometimes a 1 is not a 1! and other classics
- while and do-while loops
- increment and decrement operators
  - ++, --

```java
int num = 0, prod = 1; String numStr;
do {
    numStr = JOptionPane.showInputDialog(null,"Enter a number greater than 0: ");
    num = Integer.parseInt(numStr);
    if (num < 0) {
        System.out.println("Thanks for playing. Bye!");
    }
    else if (num == 0) {
        System.out.println("Product is zero. Exiting.");
    }
    else {
        prod = prod * num;
        if (prod > 5000) {
            System.out.println("Product exceeds 5000. Exiting.");
        }
    }
} while (num > 0 && num != 0 && prod <= 5000 );
```

Potential gotcha
- Notice that <condition> is a bit complex
  - we have multiple stop conditions
- Q: How can we clean this up?
- A: Use a flag to indicate when we should exit the loop
  - flag = boolean variable
```java
boolean continue = true; int num = 0, prod = 1; String numStr;
do {
    numStr = JOptionPane.showInputDialog(null, "Enter a number greater than zero!");
    num = Integer.parseInt(numStr);
    if (num < 0) {
        System.out.println("Thanks for playing. Bye!");
        continue = false;
    } else if (num == 0) {
        System.out.println("Product is zero. Exiting.");
        continue = false;
    } else {
        prod = prod * num;
        if (prod > 5000) {
            System.out.println("Product exceeds 5000. Exiting.");
            continue = false;
        }
    }
} while (continue);
```

### Half-loops

- Sometimes, we want to exit a loop in the middle:
  - a variable changes partway through that might negatively affect the outcome of the loop, or result in an error
  - we want to do all input handling within the loop, without pre-setting anything (and thus retyping some code)
- while and do-while test at the beginning and end of the loop, respectively
- Q: How can we do this?

### Example

```java
double rate; String rateStr;
while (true) {
    rateStr = JOptionPane.showInputDialog(null, "Enter interest rate: ");
    rate = Double.parseDouble(rateStr);
    if (rate >= 0 || rate <= 100)
        break;
    else
        JOptionPane.showMessageDialog(null, "You must enter a rate between 0 and 100.");
}
```

### Notes

- `<condition>` is always true
  - indicates that we should always run this loop
  - necessary to have a `break` statement in this case
  - watch for the infinite loop!
  - alternately, we could have used another statement that is always true, like
    ```java
    while (true) { ... }
    ```
Count-controlled loops, revisited

- So far, we've used **while** loops and **do-while** loops for both count-controlled and sentinel-controlled loops
- It would be nice if there were an easier way to implement count-controlled loops than this:

```java
int count = 0, sum = 0;
while (count < 10) {
    sum = sum + count;
    count++;
}
```

**for** loop

- Used for count-controlled loops
- The counter is handled by the loop itself, and is part of the loop condition
- A **control variable** keeps track of how many times the loop has executed
- “For these values of i, do the following...”

**Examples**

```java
int sum = 0;
for (int i=0; i<10; i++) {
    sum = sum + i;
}
for (int j=100; j>=0; j--) {
    System.out.print(j + " ");
}
int k;
for (k=1; k<200; k*2) {
    System.out.print(k + " ");
}
```

**Format**

```java
for (<initialization>; <boolean expression>; <increment>){
    <statement>
}
```
- **<initialization>**: initializes the control variable
- **<boolean expression>**: some test of the control variable
- **<increment>**: change the value of the control variable by this amount each time through
Notes

- control variable can be declared inside or outside of <initialization>
  - for (int i=0; ...) is the same as int i;
    for (i=0; ...)
- Loop will execute as long as <boolean expression> is true
- <increment> can be any change to the variable
  - adding 1, adding 3, subtracting 1, subtracting 10,
    multiplying by 4, dividing by 7, ...

Nested for loops

- Just as we nested if statements, we can nest for statements
- (we can nest while and do-while too)
- Example: looping through a two-dimensional structure
  - table
  - matrix

Example: Table

- Table contains student grades
- Grader made a mistake on everyone's grade on every assignment
- Need to raise everyone's grade by 10%
- Q: how to do this?

Grade change: pseudocode

for each row in the table,
  for each item in the row,
    multiply the item by 1.10;
Grade change: code

```java
public class Grades {
    int numStudents;
    int numHW;
    double[][] gradeList;
    public Grades() {
        this(0, 0);
    }
    public Grades(int students, int hw) {
        numStudents = students;
        numHW = hw;
        // create an array; rows correspond to students,
        // columns correspond to assignments
        gradeList = new double[numStudents][numHW];
    }
    public void populateGradeList() {
        ... // fill the table with the students' grades
    }
    public void changeGrades(int percent) {
        double factor = percent/100.0 + 1.0;
        for (int i=0; i<numStudents; i++) {
            for (int j=0; j<numHW; j++) {
                gradeList[i][j] = gradeList[i][j] * factor;
            }
        }
    }
}
```

// continued on next page

Grade change: code

```java
public void populateGradeList() {
    ... // fill the table with the students' grades
}
public void changeGrades(int percent) {
    double factor = percent/100.0 + 1.0;
    for (int i=0; i<numStudents; i++) {
        for (int j=0; j<numHW; j++) {
            gradeList[i][j] = gradeList[i][j] * factor;
        }
    }
}
```

// end of class definition

Loop summary

- Use `while` loops when you want to test the condition before executing the loop (pre-test)
- Use `do-while` loops if you want the loop to execute at least once (post-test)
- Use `for` loops if you want to execute the loop a set number of times (count-controlled)
- Watch out for infinite loops!

Creating and storing more than one object

- Up to this point, we've created and used one (or a few) objects at a time
- Sometimes, we need to create and use many objects of a particular type
  - creating and referring to each object separately is tedious and error-prone!
- Solution: use an array
Arrays

- Allow for storage of data in one or more dimensions
  - lists
  - tables
  - more complicated relationships (3+ dimensions)
- Can be used to store any type of object

Creating an array in Java

- One-dimension:
  
  `<data type>[] = new <data type>[<size>]`

- Multiple-dimension:
  
  `<data type>[][] = new <data type>[<size>][<size>]...`