

# `for` Loops

GEEN163

*“They have computers, and they may have other weapons of mass destruction.”*

**Janet Reno**

# Scanner with Files

- When you create an object of the Scanner class, you can specify a File object instead of System.in

```
java.io.File bat = new java.io.File("mydata.txt");  
java.util.Scanner bird = new java.util.Scanner( bat );  
  
    // read a number from the file  
double bug = bird.nextDouble();
```

# Useful Scanner Methods

- **nextInt()** – read an int
- **nextDouble()** – read a double
- **nextLine()** – read the whole line as a String
- **next()** – read the next word as a String
- **hasNext()** – true if there is more *data*
- **hasNextInt()** – true if there is another int
- **hasNextDouble()** – true if there is another double
- **close()** – close the file when done

# Scanner File Exceptions

- When you create a Scanner object using a File object, it could throw an exception
- Your program **must** handle the possible exception
- The easiest way to do this is to throw the exception

```
public static void main( String[] unused ) throws  
    java.io.IOException {
```

# Data You Don't Want

- Sometimes the data file has more data than you want
- If there are five numbers on a line and you only want numbers 2 and 4, it may be easiest to read all five numbers and ignore the ones you don't need
- The Scanner method `nextLine()` will read the rest of the line

# Skipping Data

- Imagine a data file looks like:

92 3/15/2009 14.5 Fred Smith

47 7/4/1953 3.75 Mary Jones

32 4/1/2013 42.1 Mike J. Snodgrass

- You need the first and third numbers on a line

```
int num1 = infile.nextInt();
```

```
infile.next();
```

```
double num2 = infile.nextDouble();
```

```
infile.nextLine();
```

# Looping Structures

- Java provides three different ways to create a loop

**while** loops

**do while** loops

**for** loops

- All of them cause the program to repeat a set of statements



# While Loop Format

- A while loop has a loop condition and a body.
- The body is repeated while the loop condition is true.

```
while (loop condition) {  
    // loop body  
}
```

# while Loop

```
double sum = 0.0, term = 4.0;
double counter = 1.0, sign = 1.0;
while ( term > 0.0001) {
    term = 4.0 / counter;
    sum += sign * term;
    sign = -1.0 * sign;
    counter = counter + 2.0;
}
System.out.println( "Pi is " + sum );
```

# What is displayed?

```
int dog = 5, cat = 1;
while (dog > 0) {
    dog = dog - cat;
    cat++;
}
System.out.println( dog );
```

A. -4

B. -1

C. 0

D. 1

E. 5

# do while Syntax

- When a do while loop gets to the end, it checks the logical expression. If true, it repeats the loop

```
do {  
    // loop body  
} while (logical expression);
```

# do while Loop

```
double sum = 0.0, term;  
double counter = 1.0, sign = 1.0;  
do {  
    term = 4.0 / counter;  
    sum += sign * term;  
    sign = -1.0 * sign;  
    counter = counter + 2.0;  
} while ( term > 0.0001);  
System.out.println("Pi is " + sum );
```

# Always Once

- A do while loop is always executed at least once

```
int cow = 5, bull = 7;
```

```
while (cow < 4 ) {
```

```
    cow = cow * cow; // never executed
```

```
}
```

```
do {
```

```
    bull = bull* bull; // executed once
```

```
} while( bull < 4 );
```

# What is displayed?

```
int dog = 5, cat = 1;  
do {  
    dog = cat - dog;  
    cat++;  
} while (dog > 0);  
System.out.println( dog );
```

A. -4

B. -1

C. 0

D. 1

E. 5

# Input Validation

- Sometimes an if and loop must test the same thing

```
do {  
    System.out.print("Enter the GPA >");  
    gpa = keyboard.nextDouble();  
    if (gpa > 4.0) {  
        System.out.println("GPA cannot be more than 4.0");  
    }  
} while (gpa > 4.0);
```



# Input Validation

- Using a boolean to avoid second comparison

```
boolean problem;
```

```
do {
```

```
    System.out.print("Enter the GPA >");
```

```
    gpa = keyboard.nextDouble();
```

```
    problem = false;
```

```
    if (gpa > 4.0) {
```

```
        System.out.println("GPA cannot be more than 4.0");
```

```
        problem = true;
```

```
    }
```

```
} while (problem);
```

# Read a File with `do while`

```
public class FileDo { // average numbers in a file
    public static void main(String[] unused)
        throws java.io.IOException {
        java.io.File dog = new
            java.io.File("numbers.txt");
        java.util.Scanner cat = new
            java.util.Scanner( dog );
        double sum = 0.0; // sum of values in file
        int count = 0;    // number of values in file
        do {
            sum += cat.nextDouble();
            count++;
        } while ( cat.hasNextDouble() );
        cat.close();
        System.out.println( sum / count );
    }
}
```

# Java **for** the Common Loop

- Java has the **for** loop that is very useful for loops that repeat a fixed number of times
- The **for** loop is equivalent to a while loop with the initialization and counter

# for Loop

- The **for** loop has three parts separated by semicolons
  - **initialization** – Done once before the loop
  - **test** – Loop executed when test is true
  - **update** – Done at the end of the loop before test

```
for ( initialization; test ; update ) {  
    // do something  
}
```

# Print the numbers 1 to 10 and squares

```
int dog, cat;  
for (dog = 1; dog <= 10; dog++) {  
    cat = dog * dog;  
    System.out.println(dog+" "+cat);  
}
```

# for and while loop equivalence

```
for ( initialization ; test; update ) {  
    statements to repeat  
}
```

is the same as

```
initialization;  
while (test) {  
    statements to repeat  
    update;  
}
```

# do while version

- To read 7 integers with a **do while** loop

```
Scanner keys = new Scanner(System.in);  
int sum = 0, count;  
count = 0;  
do {  
    sum = sum + keys.nextInt();  
    count++;  
} while(count < 7);
```

# while version

- To read 7 integers with a **while** loop

```
Scanner keys = new Scanner(System.in);  
int sum = 0, count;  
count = 0;  
while(count < 7) {  
    sum = sum + keys.nextInt();  
    count++;  
}
```



# for Example

- To read 7 integers with a **for** loop

```
Scanner keys = new Scanner(System.in) ;  
int sum = 0, count;  
for (count = 0; count < 7; count++) {  
    sum = sum + keys.nextInt() ;  
}
```

# What is displayed?

```
int elk, moose = 3;  
for ( elk = 2; elk < 4; elk++ ) {  
    moose += elk;  
}  
System.out.println(moose);
```

- A. 2
- B. 3
- C. 5
- D. 8
- E. 12

# Write this algorithm in Java with your team

- ❖ num1 = 1 and num2 = 1
- ❖ Repeat 5 times
  - print num2
  - set temp to num1 + num2
  - set num1 to num2
  - set num2 to temp

# Possible Solution

```
int num1 = 1, num2 = 1, temp, looper;  
for ( looper = 0; i < 5; i++) {  
    System.out.println( num2 );  
    temp = num1 + num2;  
    num1 = num2;  
    num2 = temp;  
}
```

# Declaring the Loop Counter

- In a **for** loop, you can declare the variable that is used as the loop counter
- The variable can only be used in the loop

```
for (int cow = 0; cow < 28; cow++) {  
    // loop body  
    // the loop counter, cow, can be used here  
}
```

*// You cannot use cow after the loop*

# What does this print?

```
int dog = 1;
for (int cat = 0; cat < 3; cat++) {
    dog = dog + cat;
}
System.out.print( dog );
```

- A. 0
- B. 1
- C. 3
- D. 4
- E. none of the above

# What does this print?

```
int dog = 1;
for (int cat = 0; cat < 3; cat++) {
    dog = dog + cat;
}
System.out.print( cat );
```

A. 0

B. 1

C. 3

D. 4

E. none of the above

# Unusual **for** Loop Example

```
int sum = 0;
for (int num = keyboard.nextInt(); num > 0 ;
      num = keyboard.nextInt() ) {
    sum = sum + num;
}
```

----- **same as** -----

```
int num = keyboard.nextInt();
while (num > 0) {
    sum = sum + num;
    num = keyboard.nextInt() ) {
}
```



# Which will loop 8 times?

- A. `for (int i = 0; i < 8; i++)`
- B. `for (int i = 0; i <= 8; i++)`
- C. `for (int i = 1; i < 8; i++)`
- D. `for (int i = 0; i <= 9; i++)`

# How useful is TuringsCraft?

- A. Very useful
- B. Useful
- C. Somewhat useful
- D. No value at all
- E. I can't log on to TuringsCraft

# How useful are copies of the slides?

- A. Very useful
- B. Useful
- C. Somewhat useful
- D. No value at all
- E. Better to have exact slides on the web

# How useful is recitation?

- A. Very useful
- B. Useful
- C. Somewhat useful
- D. No value at all
- E. I don't go to the recitation class

# How useful is the textbook?

- A. Very useful
- B. Useful
- C. Somewhat useful
- D. No value at all
- E. I do not have a textbook

# Loops in Loops

- Sometimes a program will need one loop in the body of another loop

```
int sum = 0, cat = 5, dog = 3;
for (int i = 0; i < cat; i++) {
    for (int k = 0; k < dog; k++) {
        sum += i * k;
    }
}
```

- Note that the loop counter variables (i and k) must be different

# Long Running Programs

- If a loop repeats **n** times is inside a loop that repeats **m** times, the total number of iterations of the loop is **m \* n**
- If **n** and **m** are large, it may take a very long time for the program to complete
- Loops inside loops inside loops inside loops can take a long time to run

# What does this program display?

```
int count = 0;
for (int i = 0; i < 2; i++) {
    for (int k = 0; k < 3; k++) {
        count = count + 1;
    }
}
System.out.println( count );
```

- A. 1
- B. 2
- C. 3
- D. 5
- E. 6