Motivation for Loops: Repetition

- Make the Organism speak a given number of times.
  - Where `num` is a user input into a second `speak()` method

- "Defensive Programming" considerations:
  - What if `num` was input as 0 or a negative number?
    - You should do nothing!

```java
public class Organism {
    // (lots of code omitted)

    /**
     * Organism speak its mind
     */
    public void speak() {
        System.out.println(brain.getThought());
    }

    /**
     * Organism speaks its mind
     * @param num Amount by which to speak
     */
    public void speak(int num) {
        // ???
    }
}
```
Outline

- Loops
  - While Loops
  - For Loops
- Loop Characteristics
- Nested Loops
- Conclusion
The while Loop

- **while Loops**
  - While the given boolean expression evaluates to **true**, then:
    - Perform the loop statements
    - The control loops back to verify the condition again
  - When expression evaluates **false**
    - Exit loop!

- **Syntax:**

```
while (boolean_expression) {
    //statements
}
```

*Known as the "Looping Condition"*
Tracing: (Checking Correctness)

- **Important!** To better understand looping behavior, it is useful to perform a *trace*

- A program *trace* is a table that keeps track of:
  - Values of any variables affected by loop statements
  - Value of the looping condition
  - Output to screen, if printing inside loop

<table>
<thead>
<tr>
<th>variables</th>
<th>loop cond. (true/false)</th>
<th>Return value (if applicable)</th>
<th>Output (if applicable)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Example Trace on Pig's speak() Method

```java
/**
 * Organism speaks its mind
 * @param num Amount by which to speak
 */
public void speak(int num) {
    int timesSpoken = 0;
    while (timesSpoken < num) {
        System.out.println(brain.getThought());
        timesSpoken++;
    }
}

Organism porky = new Organism();
porky.remember("I'm hangry!");
porky.speak(1);  //now trace this call
porky.speak(3);  //now trace this call
porky.speak(0);  //make sure bad inputs are handled (defensive)
porky.speak(-2); //make sure bad inputs are handled (defensive)
```

(Trace on Board)
There's a runtime bug in this method. What's wrong with this loop?

- These are generally called *"one-off" errors (VERY common)*

```java
/**
 * Organism speaks its mind
 * @param num Amount by which to speak
 */
public void speak(int num) {
    while (num >= 0) {
        System.out.println(brain.getThought());
        num--;
    }
}
```

*(Trace to see problem... and how do I fix it?)*
There's a runtime bug in this method. What's wrong with this loop?

- These are generally called *infinite loops*

```java
/**
 * Organism speaks its mind
 * @param num Amount by which to speak
 */
public void speak(int num) {
    int timesSpoken = 0;
    while (timesSpoken < num) {
        System.out.println(brain.getThought());
        timesSpoken--;
    }
}
```

*Trace to see problem... and how do I fix it?*
Example: Circle Class Revisited

- What would this mystery method do?
  - Loops that use a counter are called:
    - **Counter controlled loops**

```java
public class Circle {
    /**
     * What does this Mystery Method do?
     */
    public void mystery() {
        int i = 20;
        while (i <= 100) {
            changeSize(i);
            i++;
        }
        while (i > 20) {
            changeSize(i);
            i--;
        }
    }
}
```
Some loops don't have counters. Consider this code....

- Loops that don't depend on a counter are called:
  
  - Sentinel controlled loops (or event controlled loops)

```java
public class Circle {
    /**
     * What does this Mystery2 Method do?
     */
    public void mystery2() {
        Random rng = new Random();
        int size = -1;
        while (size != 20) {
            size = rng.nextInt(50);
            changeSize(size);
        }
    }
}
```
Code writing: Printing Even Numbers

- Print all even numbers between start (inclusive) and end (inclusive).

public void printEvens(int start, int end)

- Output when used:

  printEvens(-4, 2);
  > -4
  > -2
  > 0
  > 2

  printEvens(4, 800);
  > 4
  > 6
  > 8
  > ... (lots of lines skipped)
  > 798
  > 800

- Don't forget to check *edge cases!*

  // same start and end?
  printEvens(12, 12);
  > 12

  // start is greater than end?
  printEvens(18, 0);
  > no output

  // works with odd inputs?
  printEvens(3, 7);
  > 4
  > 6
Writing a Loop to Compute a Sum

- Write a method `sum` that inputs \( n \) (assume \( n > 0 \)) and computes and returns \( 1 + 2 + 3 + \ldots + (n - 1) + n \)

```java
/**
 * This method returns the sum from 1 to n.
 * @param n A positive integer
 * @return The sum from 1 to n
 */
public int sum(int n) {

}
```
Solution

```java
/**
 * This method returns the sum from 1 to n.
 * @param n A positive integer
 * @return The sum from 1 to n
 */
public int sum(int n) {
    int sum = 0; // holds the sum
    int current = 1; // loop counter (or, the current number to add)

    while (current <= n) {
        sum += current;
        current++;
    }

    return sum;
}
```
Outline

- Loops
  - While Loops
  - For Loops
- Loop Characteristics
- Nested Loops
- Conclusion
The for Loop

- For-Loops
  - Equivalent in expressive power to while loops
  - **initialize** statement runs before loop begins
    - Used to declare/initiate local variables used in loop body
  - **progress** statement runs after each iteration
    - Multiple statements can be separated with a comma

- Syntax:

```java
for (initialize; booleanExpression; progress) {
    loop statements
}
```
Sum Example Revisited

- We'll do the previous program using a for-loop
- Example (again): Compute the sum of 1 to n, for \( n > 1 \)

```java
public int sum(int n) {
    int sum = 0, i = 1;
    while (i <= n) {
        sum += i;
        i++;
    }
    return sum;
}
```

**while-loop version**

```java
public int sum(int n) {
    int sum = 0;
    for (int i = 1; i <= n; i++) {
        sum += i;
    }
    return sum;
}
```

**for-loop version**

[Flowchart diagram]
Finally time for us to understand the `slowMoveHorizontal` method:

```java
public void slowMoveHorizontal(int distance)
{
    int delta;

    if (distance < 0)
    {
        delta = -1;
        distance = -distance;
    }
    else
    {
        delta = 1;
    }

    for (int i = 0; i < distance; i++)
    {
        xPosition += delta;
        draw();
    }
}
```
Example: Computing Factorials

- The factorial of an integer \( n \ (n \geq 0) \) is defined:

\[
 n! = n \times (n - 1) \times (n - 2) \times \ldots \times 2 \times 1
\]

- *Note that* \( n! = 1 \) *when* \( n \) *is 0*

```java
public int factorial(int n) {
    if (n == 0) {
        return 1;
    }

    int fac = n;
    for (int i = n-1; i >= 2; i--) {
        fac *= i;
    }
    return fac;
}
```
Another Example: Turtle Class Revisited

- Next Lab: Turtle Graphics!
- Here's Turtle's public API

<table>
<thead>
<tr>
<th>Signature</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>public Turtle()</td>
<td>Constructor to create a Turtle object</td>
</tr>
<tr>
<td>public void forward(double distance)</td>
<td>Move turtle forward by specified distance</td>
</tr>
<tr>
<td>public void penDown()</td>
<td>Lower the pen to draw</td>
</tr>
<tr>
<td>public void penUp()</td>
<td>Lift pen up</td>
</tr>
<tr>
<td>public void left(double angle)</td>
<td>Turn turtle left by given angle</td>
</tr>
<tr>
<td>public void right(double angle)</td>
<td>Turn turtle right by given angle</td>
</tr>
<tr>
<td>public void setPenColor(String color)</td>
<td>Sets the color of the turtle's pen.</td>
</tr>
</tbody>
</table>
public class Drawing {
    private Turtle pen;

    /**
     * Constructor to create a new Turtle Graphics object
     */
    public TurtleDrawer() {
        pen = new Turtle();

        //other initialization code I'm withholding
        // (code omitted)
    }

    /**
     * Draws a square
     * @param sideLength length of each side
     */
    public void drawSquare(int sideLength) {
        //code...
    }
}
Implementing `drawSquare()`

- Steps:
  - Lower the pen
  - Create a loop that runs four times
    - Draw an edge, turn by 90 degrees
    - Repeat

```java
public void drawSquare(int sideLength)
{
    pen.penDown(); //get pen ready to draw

    //draw four edges
    for (int sidesDrawn = 0; sidesDrawn < 4; sidesDrawn++)
    {
        pen.forward(sideLength);
        pen.left(90);
    }
}
```
Drawing More Complex Shapes

- Lab 6: Last semester's students' work
Outline

- Loops
  - While Loops
  - For Loops
- Loop Characteristics
- Nested Loops
- Conclusion
Counter-Controlled Loops

- Loop conditions are often controlled by a counter variable
  - These loops are known as *counter-controlled loops*
    - Generally, for-loops are used
  - Examples:

```java
public void speak(int N) {
    //speak N times
    for (int i = N; i > 0; i--) {
        speak();
    }
}
```

- Why? Counter-controlled loops have definite repetition: the number of iterations is known before loop begins execution
Sentinel-Controlled Loops

- When loop conditions are controlled by an event occurring
  - These are known as *sentinel-controlled* (or *event-controlled*) loops
    - Generally, while-loops or do-while loops are used

- Sentinel-controlled loops have *indefinite repetition*: the number of iterations is unknown to the programmer

```java
while (hairNotClean) {
    shampoo();
    lather();
    rinse();
}
```

```java
for (; hairNotClean;) {
    shampoo();
    lather();
    rinse();
}
```
Say we have a Die class with a method that rolls 1 through 6 randomly.

```java
public class Die {
    private int faceValue;
    private Random rng;

    public Die() {
        faceValue = 1;
        rng = new Random();
    }

    public int getFaceValue() {
        return faceValue;
    }

    public void roll() {
        faceValue = rng.nextInt(6) + 1;
    }
}
```

Write a class that holds two dice, and a method rolls the dice repeatedly until they add up to 2, i.e., each die shows 1.
Sentinel Loops (Cont.)

- Why is the looping condition called an event ("sentinel")?
  - You don't know when total will ever reach 2!

```java
class DicePair {
    private Die d1;
    private Die d2;

    public DicePair() {
        d1 = new Die();
        d2 = new Die();
    }
}
```
Sentinel Loops (Cont.)

- Why is the looping condition called an *event* ("sentinel")?
  - You don't know *when* total will ever reach 2!

```java
public class DicePair {
    private Die d1;
    private Die d2;

    public DicePair() {
        d1 = new Die();
        d2 = new Die();
    }

    /** @return number of rolls needed to get snake eyes */
    public int snakeEyes() {
        int sumRolls, numRolls;
        d1.roll();
        d2.roll();
        sumRolls = d1.getFaceValue() + d2.getFaceValue();
        numRolls++;
        while (sumRolls != 2) {
            d1.roll();
            d2.roll();
            sumRolls = d1.getFaceValue() + d2.getFaceValue();
            numRolls++;
        }
        return numRolls;
    }
}
```

But isn't this just one iteration of the loop that's executed unconditionally?
Outline

- Loops
  - While Loops
  - For Loops
    - Do-while Loops
- Nested Loops
- Scanner Class: Interacting with User Input
- Conclusion
Do-While Loops

So far, we've seen while-loops and for-loops

- do-while loops

```java
//prior statements
do {
    //loop statements
} while (cond);
//remainder statements
```

Expressive power is also equivalent to while-loop and for-loop

- Loop statements are done before the loop condition is checked
- Therefore, guarantees at least one iteration is executed
Sentinel Loops (Cont.)

- Same method now cleaner with a do-while loop

```java
public class DicePair {
    private Die d1;
    private Die d2;

    public DicePair() {
        d1 = new Die();
        d2 = new Die();
    }

    /** @return number of rolls needed to get snake eyes */
    public int snakeEyes() {
        int sumRolls = 0, numRolls = 0;
        do {
            d1.roll();
            d2.roll();
            sumRolls = d1.getFaceValue() + d2.getFaceValue();
            numRolls++;
        } while (sumRolls != 2); //don't know when this event will happen
        return numRolls;
    }
}
```
Outline

- Loops
  - While Loops
  - For Loops
- Loop Characteristics
- Nested Loops
- Conclusion
Nested Loops

- Loops can contain any valid Java statement, which includes another loop.
  - Called "nested loops"

```java
public String mystery1() {
    String str = "";
    for (int i=0; i<2; i++) {
        for (int j=0; j<2; j++) {
            str += "X";
        }
        str += "\n";
    }
    return str;
}
```

- Return value: ?
Another Nested-Loop Example

- Apply a trace for when `mystery2(4)` is called

```java
public void mystery2(int n) {
    String str = "";
    for (int i = 0; i < n; i++) {
        for (int j = 0; j <= i; j++) {
            str += "*";
        }
        System.out.println(str);
        str = "";
    }
}
```

- Which is the correct output?

A

```
****
***
** *
```

B

```
*
***
*****
```

C

```
****
*****
*****
```

D

```
*
**
***
```

E

```
****
***
**
```
How Do I Get Output E?

What change(s) do I need to make to get output E?

```java
public void mystery2(int n) {
    String str = "";
    for (int i = 0; i < n; i++) {
        for (int j = 0; j <= i; j++) {
            str += "*";
        }
        System.out.println(str);
        str = "";
    }
}
```

Which is the correct output?

- A
- B
- C
- D
- E
What change(s) do I need to make to get outputs A and B?

```java
public void mystery2(int n) {
    String str = "";
    for (int i = 0; i < n; i++) {
        for (int j = 0; j <= i; j++) {
            str += "*";
        }
        System.out.println(str);
        str = "";
    }
}
```

Which is the correct output?

- A
- B
- C
- D
- E
More Practice with Nesting

- Try doing the following on your own time:

```java
/**
 * A method that prints a checkered pattern
 * @param len width and height
 */
public void checker(int len)
```

- `checker(7)` prints this to the terminal:
public void drawMyPattern()
{
    double triangleSize = 50;
    int numRows = 3, numCols = 4;

    // draws a 3 (row) by 4 (column) wall of triangles
    for (int i = 0; i < numRows; i++)
    {
        for (int j = 0; j < numCols; j++)
        {
            // draw one triangle
            drawPolygon(triangleSize, 3, "red");

            // move pen in position to draw next triangle
            pen.penUp();
            pen.forward(triangleSize);
            pen.penDown();
        }

        // move turtle in position for next row
        pen.penUp();
        pen.left(90);
        pen.forward(triangleSize);
        pen.left(90);
        pen.forward(triangleSize*numCols);
        pen.right(180);
    }
}
More Problems

- Write a nested loop that would output the following

  A) `public void revealer(int n)`

     - When `n == 7`:
       1******
       12*****
       123****
       1234***
       12345**
       123456*
       1234567

  B) `public void printTriangle(int n)`

     - When `n == 4`:
       *
       **
       ****
       *****

  C) `public void printFactors(int n)`

     - When `n == 150`:
       2
       3
       5
       5
     - When `n == 11`:
       11
     - When `n == 70`:
       2
       5
       7
Fast repetition is one of the main reasons we built computers in the first place!

- Computers are powerful! Millions of loop iterations per second
- Computers are loyal! Won't stop and complain
- Computers are stupid!

```c
// This is an infinite loop
int i = 0;
while (i < 1)
{
    // stuff
    i--;
}
```
Computers are powerful!

When designing loops, follow these rules of thumb:

- What code needs to be repeated (e.g., loop body)?
- How many repetitions (i.e., what is the looping condition)?
- How is progress being made to terminate the loop?

Which loop to use?

- Use for-loops when you know the number of iterations beforehand
- Use while-loops when you don't know the number of iterations beforehand
With loops, we can iterate through lots of variables without it ever tiring!

- How do I create (and manage) lots of variables? (*Next: Arrays and ArrayList*)
Hwk 4: Robots
  • Due tonight!

Exams 50% graded

Last time...
  • Finished Organism
  • Started code repetition with while loops
    - Basic code reading
    - Infinite loops

Today: More loops! Writing loops!
Exams graded (15% of final grade)

- Avg = 82
Hwk 5?

Lab 6 post-mortem:
- Infinite loops abound!
- What's up with drawGrid?
  - Nested loops necessary

Last time...
- Read more code
- Wrote printEvens

Today...
- For loops, do-while loops
- Nested loops