CSCI 161
Introduction to Computer Science
Motivation

- Consider our Organism class

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

    /**
     * Organism speak its mind
     */
    public void speak() {
        if (this.brain.isAwake() == true) {
            System.out.println(this.name + " says: " + this.brain.getThought());
        }
    }

    /**
     * Organism speaks its mind
     * @param num The number of times to speak
     */
    public void speak(int num) {
        // ???
    }
}
```
Outline

- Loops
  - While Loops
  - For Loops
- Loop Characteristics
- Nested Loops
- Conclusion
Imagine an if-statement that keeps repeating until false!

- While the *looping condition* evaluates *true*, then:
  - Perform the "loop statements"
  - After last statement, loops back to verify the condition again

- When expression evaluates *false*
  - Exit loop!
Tracing: (Checking Correctness)

- **Important!** 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>Printout (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>
</tbody>
</table>
Example Trace of the speak() Method

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

Organism adam = new Organism();
adam.remember("I'm hangry!");
adam.speak(3); // now trace this call
adam.speak(0); // make sure "edge cases" are handled (defensive programming)
adam.speak(-2); // another edge case

(Trace on Board)
What does the following loop print?

```java
public void mystery1() {
    int n = 1;
    while (n < 100) {
        n *= 2;
        System.out.println(n);
    }
}
```

Explain what this code returns.

```java
public String mystery2(String str, int n) {
    String ret = str;
    while (n > 0) {
        ret += str;
        n--;
    }
    return ret;
}
```
What would this mystery method do if implemented in the Circle Class?

```java
public class Circle {
    /**
     * What does this Mystery Method do?
     */
    public void mystery() {
        int i = 20;
        while (i <= 100) {
            this.changeSize(i);
            i++;
        }
        while (i > 20) {
            this.changeSize(i);
            i--;
        }
    }
}
```
Common Bugs

- Don't always have to count up by 1s.
  - This loop does the same thing as before, but counts down...
    - There's a bug though (can you find it?)

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

- These are generally called "1-off" errors
  - *(Very common when writing loops)*

```java
/**
 * Organism speaks its mind
 * @param num Amount by which to speak
 */
public void speak(int num) {
    if (this.brain.isAwake() == true) {
        while (num > 0) {  // fixed!
            System.out.println(this.brain.getThought());
            num--;
        }
    }
}
```
There's another runtime bug in 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) {
    if (this.brain.is Awake() == true) {
        while (num > 0) {
            System.out.println(this.brain.getThought());
            num++;
        }
    }
}
```
David's 4-Step Instruction for Writing Loops

1. Try writing the algorithm (in English) without loops

2. Identify the set of repeated instructions and generalize
   a. Then rewrite the algorithm by reducing the repeated segment

3. Write the loop in Java:
   a. Loop Condition? Answer this question:
      ‣ Repeat the above code segment from Step 2 "as long as..."?
        (e.g., Has the counter reached the final value?)
   b. Set up the variables for entering loop for the first time

4. Run hand traces to verify correctness
Code Writing: Print a String Vertically

- **Preliminary:** String class has a `public char charAt(int pos)` method that returns the character at position `pos`.

- Prints each character in the given string on a separate line

```java
public void printVertical(String str)
```

- Sample Output:

```java
this.printVertical("David");
```

```
D
a
v
i
d
```
Code Writing: Reversing a String

- Return the given string in reverse order

```java
public String reverse(String str)
```

- Sample Output:

```java
System.out.println(this.reverse("David"));
> "divaD"

System.out.println(this.reverse("hello world!"));
> "!dlrow olleh"

System.out.println(this.reverse(""));
> ""
```
Print all even numbers between **start** and **end**, inclusive.

```java
public void printEvens(int start, int end)
```

**Sample Output:**

```
this.printEvens(-4, 2);
> -4
> -2
> 0
> 2

this.printEvens(4, 80);
> 4
> 6
> 8
> ...
> 78
> 80
```

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

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

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

// works with odd inputs?
this.printEvens(3, 7);
> 4
> 6
```
/**
 * Prints all even integers between the given ends, inclusive.
 * @param start Starting point
 * @param end End point
 */
public void printEvens(int start, int end) {
    // Is start odd? If so, make it even
    if (start % 2 == 1) {
        start++;
    }

    // Loop from start to end, incrementing by 2
    while (start <= end) {
        System.out.println(start);
        start += 2;
    }
}
Another Example: Prime Number Verifier

- Given an integer $n$, check if $n$ is prime
  - A prime number is only evenly divisible by 1 and itself
  - A prime number is greater than 1

- Preliminaries:
  - How to check if an int divides evenly into another int?
  - Now, apply David's 4 step plan...
Given a positive number \( n \), check if \( n \) is prime.

- One solution

```java
public boolean isPrime(int n) {
    // Try all divisors in decreasing order until it reaches 1.
    int divisor = n-1;
    while (divisor > 1) {
        if (n % divisor == 0) {
            return false;
        }
        divisor--;
    }
    // If we got here, we made it through the loop. N must be prime!
    return true;
}
```

- Check *edge cases*...
  - Does the method work with \( n \) input as 0? 1? 2? -1?
Given a positive number \( n \), check if \( n \) is prime.

- Better solution

```java
public boolean isPrime(int n) {
    if (n <= 1) {
        return false;
    }

    // Try all divisors in decreasing order until it reaches 1.
    int divisor = n-1;
    while (divisor > 1) {
        if (n % divisor == 0) {
            return false;
        }
        divisor--;
    }

    // If we got here, we made it through the loop. N must be prime!
    return true;
}
```
Outline

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

- Equivalent in power to while loops
  - initialize statement runs once before we go into loop
    - Used to declare/initialize counter variables used in loop
  - progress statement runs after each iteration
    - Multiple statements can be separated with a comma

Syntax:

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

- Rewriting the `speak(int num)` method using for-loop

```java
public void speak(int num) {
    if (this.brain.isAwake() == true) {
        int timesSpoken = 0;
        while (timesSpoken < num) {
            System.out.println(this.brain.getThought());
            timesSpoken++;
        }
    }
}
```

```java
public void speak(int num) {
    if (this.brain.isAwake() == true) {
        for (int timesSpoken = 0; timesSpoken < num; timesSpoken++) {
            System.out.println(this.brain.getThought());
        }
    }
}
```

While Loop Version

For Loop Version
Example: Compute the sum from 1 to n.

- $1 + 2 + 3 + ... + (n-1) + n$

```java
public int sum(int n) {
    int sum = 0;
    for (int i = 1; i <= n; i++) {
        sum += i;
    }
    return sum;
}
```
Circle class' `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++) {
        this.xPosition += delta;
        this.draw();
    }
}
```
### Another Example: Turtle Class Revisited

- Tomorrow's Lab: Turtle Graphics!!
- Here's Turtle's API

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

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

        // other initialization code omitted
    }

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

- **Steps:**
  - Lower the pen, then,
  - Draw an edge, turn by 90 degrees
    - Repeat 4 times

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

    // draw four edges
    for (int sides = 0; sides < 4; sides++) {
        this.pen.forward(length);
        this.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
A Die Class to Exemplify

- Let's write a Die class

```java
public class Die {
    private int faceValue; // what's currently facing up?
    private Random rng;

    public Die() {
        // TODO
    }

    public void roll() {
        // TODO
    }
}
```
Loop Types: 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 for counter-controlled loops

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

- Write a method loop `rollRepeatedly(int N)` that rolls the die $N$ times.
  - Counter-controlled: Loop condition depends on a counter.
Loop Types: 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) {
    this.shampoo();
    this.lather();
    this.rinse();
}
for (; hairNotClean;) {
    this.shampoo();
    this.lather();
    this.rinse();
}
```
Write a method `rollUntil(int F)` that returns the number of rolls it took for face value `F` to turn up!

- The loop condition *doesn't depend* on a counter.

---
Aside: Do-Loops

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

```java
do {
    // loop statement 1;
    // loop statement 2;
    // ...
} while (cond);
```

- 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
Outline

- Loops
  - While Loops
  - For Loops

- Loop Characteristics

- Nested Loops

- Conclusion
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
            this.drawPolygon(triangleSize, 3, "red");

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

        //move turtle in position for next row
        this.pen.penUp();
        this.pen.left(90);
        this.pen.forward(triangleSize);
        this.pen.left(90);
        this.pen.forward(triangleSize*numCols);
        this.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
    - 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--;
}
```
Conclusion (Cont.)

- 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)*
Reminders:

- Exam Thursday (review Wednesday)
  - Things to bring:
    - 1 page cheat-sheet, front/back okay
    - String API handout
    - Calculator allowed; no phones please
- Homework 3 due Wednesday

Last time...

- Worked with the String API
- Know how to manipulate Strings: break apart, put together, search, etc.
Exam 1 post-mortem
  • Review

Last time...
  • Repetition with while loops

Today:
  • More examples of while loops
  • For loops (if time permits)
Exam 1 graded: Max = 94.5, Avg = 79

- Weak points:
  - Data types, particularly, String operations
  - Generating a random number in range [low, high]
Administrivia (Cont)

- Reminder:
  - Hwk 4: Robots due Friday 3/6

- Hwk 5 (Loops): Due next Friday 3/13

- Last time...
  - Reviewed solutions to Exam 1

- Today:
  - More examples of while loops
  - For loops (if time permits)
Reminders:
- Hwk 4: Robots due tonight
- Hwk 5 (Loops): Due next Friday 3/13
- Lab 6 post mortem
  - Submit your drawings to me!

Last time
- More examples of while loops
- For loops

Today:
- Choosing loop types, do-loops, and nesting loops