CSCI 161
Introduction to Computer Science

Est. 1888
UNIVERSITY of
PUGET
SOUND

Department of Mathematics
and Computer Science

Lecture 6
Loops

Motivation: What's Wrong with This Code?

- Print the name of your school 6 times:

```
System.out.println("Tunghai");
System.out.println("Tunghai");
System.out.println("Tunghai");
System.out.println("Tunghai");
System.out.println("Tunghai");
System.out.println("Tunghai");
```

- This code runs correctly. Thinking *ahead*... what's wrong with this code?
  - It's tedious
  - It's error-prone (How many times did you print?)
  - It's not "scalable" (What if you wanted to print 100 times? 500 times? Forever?)

The While Loop

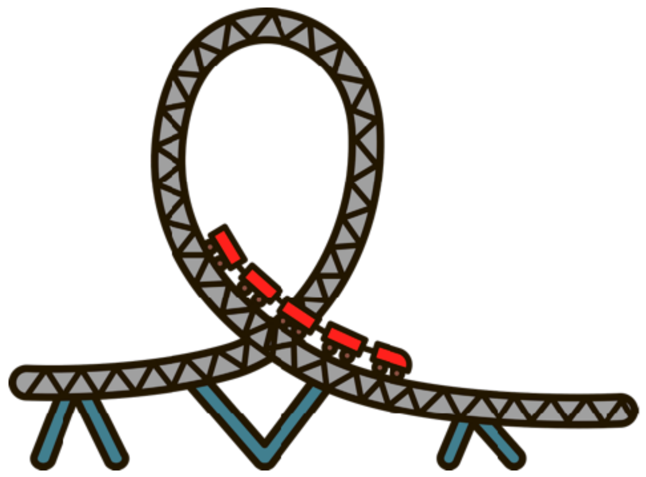
- *Code Repetition ("loops")* are among the most important constructs in programming.
  - "Computers don't tire out"
  - Loops offer crazy amounts of expressive power for programmers!

- The *while-loop* syntax:

```
while (some-boolean-loop-condition) {

  // statement
  // statement
  // ...

}
```



The While Loop (2)

- *Code Repetition ("loops")* are among the most important constructs in programming.
  - "Computers don't tire out"
  - Loops offer crazy amounts of expressive power for programmers!

- Rewriting the original code using a while loop:

```java
// print "Tunghai" 6 times
int count = 0;
while (count < 6) {
   System.out.println("Tunghai");
   count++;
}
```

Code Reading Example

- What would this mystery Circle method do?

```
public class Circle {

    /**
     * What does this Mystery Method do?
     */
    public void mysteryLoop() {
       int i = 20;
       while (i <= 100) {
            changeSize(i);
            i++;
       }
       while (i > 20) {
            changeSize(i);
            i--;
       }
    }
}
```

Code Reading Examples (2)

- 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 = "";
   while (n > 0) {
      ret += str;
      n--;
   }
   return ret;
}
```

Tracing (Understanding the Loop Execution)

- **Important!** A program *trace* is a table that keeps track of:
  - A list of all variables affected by the loop statements
  - The loop condition
  - Output to the screen, if printing inside loop
  - Return value, if returning something
  - Each row is corresponds to an iteration

variables	loop cond. (true/false)	Printout (if applicable)	Return value (if applicable)

Code Tracing Example (2)

- What does this loop do?

```java
int i = 0;
while (i < 4) {
   System.out.println(i * i);
   System.out.println((i+1) * (i+1));
   i++;
}
```

- Loop trace

Simple While Loop Example (Solution)

- What does this loop do?

```java
int i = 0;
while (i < 4) {
   System.out.println(i * i);
   System.out.println((i+1) * (i+1));
   i++;
}
```

- Output:

```
0
1
1
4
4
9
9
16
```

- Loop trace

variables	loop cond. (true/false)	Print Out
i		
0	TRUE (go!)	0 1
1	TRUE (go!)	1 4
2	TRUE (go!)	4 9
3	TRUE (go!)	9 16
4	FALSE (stop!)	

Common Bugs

- This loop should print **"bug"** exactly ***num*** times
  - There's a bug in this loop. Find it!

```
public void loopBug(int num) {
   while (num >= 0) {
       System.out.println("bug");
       num--;
   }
}
```

OOOPs! You get an extra iteration!

Common Bugs (Fixed)

- You get one more iteration!
- These are generally called ***"1-off** bugs"*
  - *(Very common when writing loops -- test for 1-offs often!)*

```java
public void loopBug(int num) {
   while (num > 0) {  // wanted num > 0, not num >= 0
       System.out.println("bug");
       num--;
   }
}
```

Common Bugs (2)

- Again, this loop should print "Hello World" num times.
  - There's another runtime bug in this loop.

```
public void bigBug(int num) {
    while (num > 0) {
            System.out.println("Hello World!");
            num++;
    }
}
```

Bug -- No progress toward loop termination:
The counter is increasing, and the loop condition runs if counter is positive!

Code Tracing Exercises

- Explain why the following code fragment is an infinite-loop.

```
 /** Print 10, 20, 30, 40, 50 */
public void mystery3() {
   int n = 1;
   while (n != 50) {
      n += 10;
      System.out.println(n-1);
   }
}
```

Code Tracing Exercises

- Explain why the following code fragment is an infinite-loop.

```java
 /** Print 10, 20, 30, 40, 50 */
public void mystery3() {
   int n = 1;
   while (n != 50) {
       n += 10;
       System.out.println(n-1);
   }
}
```

Soln:
Terminating condition: loop stops when **n** hits 50
But **n** starts at 1, and increases by 10 each iteration

- Fix:

```java
 /** Print 10, 20, 30, 40, 50 */
public void mystery3() {
   int n = 10;  // the counter's initial value was wrong!!
   while (n != 50) {
       n += 10;
       System.out.println(n);
   }
}
```

While Loops

- Recall the while loop syntax

```
while (some-boolean-loop-condition) {
  // loop statements
}
```

- Our Goal:
  - Express repeated blocks of code using the **while** syntax
  - Be careful to do *no more* and *no fewer* iterations than intended
    - Test, test, test for various "edge" cases

Code Writing: Printing Even Numbers

- Print all even numbers between **start** and **end**, inclusive.

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

- Sample Output:

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

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

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

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

// works with odd inputs?
printEvens(3, 8);
> 4
> 6
> 8

// start is given larger than end?
printEvens(4, 0);
> 0
> 2
> 4
```

Printing Even Numbers (Soln)

```
/**
 * Prints all even integers between the given ends, inclusive.
 * @param start Starting point
 * @param end End point
 */

public void printEvens(int start, int end) {
   // swap start and end if they were given in the wrong order
   if (start > end) {
       int tmp = start;
       start = end;
       end = start;
   }

   // Is start odd? If so, make it even by adding 1 to it
   if (start % 2 == 1) {
      start++;
   }

   // Loop from start to end, incrementing by 2
   while (start <= end) {
      System.out.println(start);
      start += 2;
   }
}
```

Code Writing: Print a String Vertically

- **Recall:** The String class has a `public char charAt(int pos)` method that returns the character at position *pos*.
- Write a method to print each character in the given string on a separate line

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

- Sample Output:

```
printVertical("David");

D
a
v
i
d
```

```
printVertical("Tunghai");

P
u
g
e
t

S
o
u
n
d
```

Solution

```java
/**
 * Prints a string vertically.
 * @param str any given string
 */
 public void printVertical(String str) {
   int i = 0;
   while (i < str.length()) {
      System.out.println(str.charAt(i));
      i++;
   }
}
```

Print a String in Reverse and Vertically

- **Now** Write a method to print each character in the given string on a separate line, but in reverse order

```
public void printRevVertical(String str)
```

- Sample Output:

```
printRevVertical("David");

d
i
v
a
D
```

```
printRevVertical("Tunghai");

d
n
u
o
S

t
e
g
u
P
```

Solution

```
/**
 * Prints a string in reverse vertically.
 * @param str any given string
 */
 public void printRevVertical(String str) {
   int i = str.length() - 1;
   while (i >= 0) {
      System.out.println(str.charAt(i));
      i--;
   }
}
```

Outline

- Loops
  - While Loops
  - For Loops
  - Approach to Writing Loops (Event Controlled)
- Nested Loops
- Conclusion

for Loops

- *For-Loops* are used to simplify writing counter-controlled loops
  - Initialization statement runs right before the attempt to enter the loop
    - Skipped in all subsequent iterations
  - Progress statement runs unconditionally *after the loop body of statements*
    - Makes changes to the counter

- For-Loop Syntax:

```
for (counter initialization; loop condition; counter progress) {

    // loop statements

}
```

Revisiting Previous Examples

- Use a **for-loop**! Given an integer value N, print the next 5 values.

```
public void next5(int N)
```

- Compare **while-loop** versus **for-loop**: They *both* work!

```
// while loop version
public void next5(int N) {
   int x = 1;
   while (x <= 5) {
      System.out.println((n + x));
      x++;
   }
}
```

```
// for loop version
public void next5(int N) {
  for (int x = 1; x <= 5; x++) {
      System.out.println((n + x));
   }
}
```

Code Writing: Print a String Vertically

- **Recall:** The 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:

```
printVertical("David");

D
a
v
i
d
```

Revisiting Previous Examples

- Use a **for-loop**! Print a given string vertically

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

- Solution

```
// for loop version
public void printVertical(String str) {

  for (int i = 0; i < str.length(); i++) {
      System.out.println(str.charAt(i));
   }

}
```

For-Loop Writing: Reversing a String

- Return the given string in reverse order

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

- Sample Output:

```
System.out.println(reverse("david"));
> "divad"

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

System.out.println(reverse(""));
> ""
```

Reversing a String (Solution)

- Return the given string in reverse order

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

    // the string to be returned
    String rev = "";

    // traverse the string in reverse order to
    // build up the new string
    for (int i = str.length(); i >= 0; i--) {
        rev += str.charAt(i);
    }

    return rev;
}
```