Lecture 6: While Loops and the Math Class

Building Java Programs: A Back to Basic Approach

by Stuart Reges and Marty Stepp

`while` loops

Categories of loops

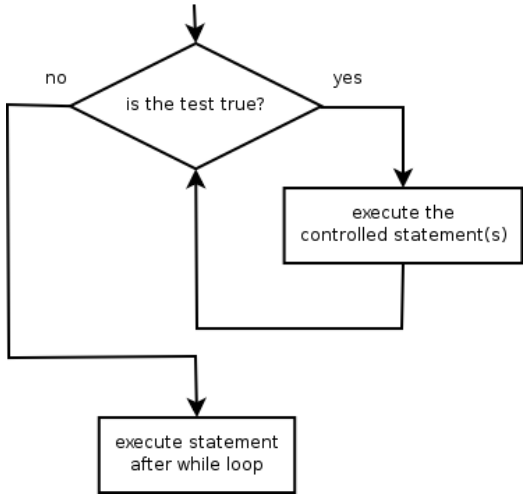
- **definite loop**: Executes a known number of times.
  - The `for` loops we have seen are definite loops.
    - Print "hello" 10 times.
    - Find all the prime numbers up to an integer *n*.
    - Print each odd number between 5 and 127.

- **indefinite loop**: One where the number of times its body repeats is not known in advance.
    - Prompt the user until they type a non-negative number.
    - Print random numbers until a prime number is printed.
    - Repeat until the user has types "q" to quit.

The `while` loop

- **`while` loop**: Repeatedly executes its body as long as a logical test is true.

```
while (test) {
    statement(s);
}
```



- Example:

```
int num = 1;                          // initialization
while (num <= 200) {                  // test
    System.out.print(num + " ");
    num = num * 2;                    // update
}

// output:  1 2 4 8 16 32 64 128
```

Example `while` loop

```
// finds the first factor of 91, other than 1
int n = 91;
int factor = 2;
while (n % factor != 0) {
    factor++;
}
System.out.println("First factor is " + factor);

// output:  First factor is 7
```

- `while` is better than `for` because we don't know how many times we will need to increment to find the factor.

What's wrong?

```
int count = 10;

while(count > 9)
{
  count++;
}
```

– Infinite loop!!

Corrected

```
int count = 10;

while(count < 13)
  System.out.println(count + " ");
  count++;
```

– Still infinite loop!!

Finally Corrected

```
int count = 10;

while(count < 13)
{
  System.out.println(count + " ");
  count++;
}
```

– Correct!

Curly braces {}

- Curly braces mark the body of methods, for loops and conditional blocks. They are not necessary if the body or the block consists of only one statement.

- The following are equivalent.

```
for (int i = 5; i <= 25; i+=1) {
        System.out.print(i + ” “);
}
```

```
for (int i = 5; i <= 25; i+=1)
        System.out.print(i + ” “);
```

Curly braces {}

- The following are also equivalent.

```
if (x <= 1) {
        System.out.print(x + ” “);
}
```

```
if (x <= 1)
        Ssytem.out.print(x + ” “);
```

Curly braces {}

- The following are NOT equivalent.

```
int sum = 0;
for (int i = 1; i <= 10; i+=1) {
        System.out.print(i + " ");
        sum += i;
}

int sum = 0;
for (int i = 5; i <= 25; i+=1)
        System.out.print(i + " ");
        sum += i;    //error, why?
```

The Math Class

Static Methods

The Math class has many useful **static** methods. To call a static method from a different class than the current point of your code, use the following syntax.

```
Math.methodName(parameters);
```

```
double answer = Math.sqrt(9.2);
double a = Math.sin(3.67);
int b = Math.round(5.6755);
```

Static Method Calling

Calling a **static** method from a different class.

```
public class Math{
        public static int abs(int x)
        {…}
        public static double sqrt(double x)
        {…}
}
public class MyClass{
        public static void main(String[] args)
        {
                System.out.println(Math.abs(-5));
                double y=Math.sqrt(9.2);
        }
}
```

Java's `Math` class

Method name	Description
`Math.abs(`*value*`)`	absolute value
`Math.ceil(`*value*`)`	rounds up
`Math.floor(`*value*`)`	rounds down
`Math.log10(`*value*`)`	logarithm, base 10
`Math.max(`*value1*`, `*value2*`)`	larger of two values
`Math.min(`*value1*`, `*value2*`)`	smaller of two values
`Math.pow(`*base*`, `*exp*`)`	*base* to the *exp* power
`Math.random()`	random `double` between 0 and 1
`Math.round(`*value*`)`	nearest whole number
`Math.sqrt(`*value*`)`	square root

`Math.sin(`*value*`)`
`Math.cos(`*value*`)`
`Math.tan(`*value*`)`
	sine/cosine/tangent of an angle in radians

`Math.toDegrees(`*value*`)` `Math.toRadians(`*value*`)`	convert degrees to radians and back

Constant	Description
`Math.E`	2.7182818...
`Math.PI`	3.1415926...

Calling `Math` methods

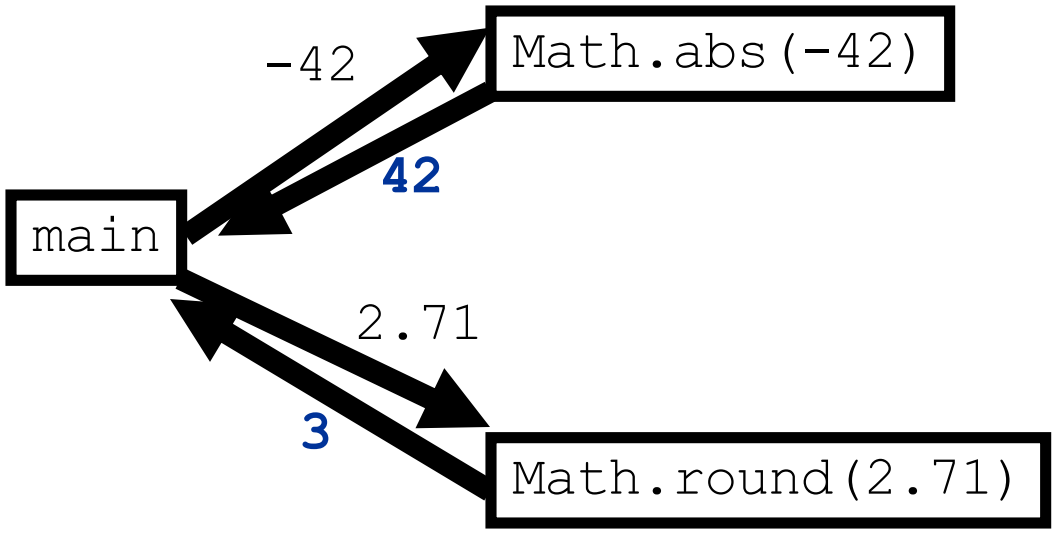
- Examples:

```
double squareRoot = Math.sqrt(121.0);
System.out.println(squareRoot);              // 11.0

int absoluteValue = Math.abs(-50);
System.out.println(absoluteValue);           // 50

System.out.println(Math.min(3, 7) + 2);      // 5
```

- The `Math` methods do not print to the console.
  - Each method produces ("returns") a numeric result.
  - The results are used as expressions (printed, stored, etc.).

Return

- **return**: To send out a value as the result of a method.
  - The opposite of a parameter:
    - Parameters send information *in* from the caller to the method.
    - Return values send information *out* from a method to its caller.
      - A call to the method can be used as part of an expression.



Math questions

- Evaluate the following expressions:
  - `Math.abs(-1.23)`
  - `Math.pow(3, 2)`
  - `Math.pow(10, -2)`
  - `Math.sqrt(121.0) - Math.sqrt(256.0)`
  - `Math.round(Math.PI) + Math.round(Math.E)`
  - `Math.ceil(6.022) + Math.floor(15.9994)`
  - `Math.abs(Math.min(-3, -5))`

- `Math.max` and `Math.min` can be used to bound numbers.

  Consider an `int` variable named `age`.
  - What statement would replace negative ages with 0?
  - What statement would cap the maximum age to 40?

Math questions

Write a method `withinHalf` which takes two double parameters and return true if they are within .5 of each other and false otherwise.

```
withinHalf(4,5.1) // returns false
withinHalf(3.4,3.9) // returns true
withinHalf(3.9,3.4) // returns true
withinHalf(-1.2,-1.1) // returns true
```

```
public static boolean withHalf(double x, double y)
{
    return Math.abs(x - y) <= .5;
}
```

Quirks of real numbers

- Some `Math` methods return `double` or other non-`int` types.

```
int x = Math.pow(10, 3);   // ERROR: incompat. types
```

- Some `double` values print poorly (too many digits).

```
double result = 1.0 / 3.0;
System.out.println(result);    // 0.3333333333333
```

- The computer represents `double`s in an imprecise way.

```
System.out.println(0.1 + 0.2);
```

  - Instead of 0.3, the output is `0.30000000000000004`

Type casting

- **type cast**: A conversion from one type to another.
  - To promote an `int` into a `double` to get exact division from `/`
  - To truncate a `double` from a real number to an integer

- Syntax:

```
(type) expression
```

Examples:

```
double result = (double) 19 / 5;      // 3.8
int result2 = (int) result;           // 3
int x = (int) Math.pow(10, 3);        // 1000
```

More about type casting

- Type casting has high precedence and only casts the item immediately next to it.
  - `double x = (double) 1 + 1 / 2;  // 1.0`
  - `double y = 1 + (double) 1 / 2;  // 1.5`
- You can use parentheses to force evaluation order.
  - `double average = (double) (a + b + c) / 3;`
- A conversion to `double` can be achieved in other ways.
  - `double average = 1.0 * (a + b + c) / 3;`

Random Numbers

Random numbers

- Math.random() produces a number from 0(inclusive) to 1 exclusive.
  - `double x = Math.random(); // 0.0 <= x <1.0`
  - `double x = 3*Math.random(); // 0.0 <= x< 3.0`
  - `double x = Math.random()+2; // 2.0 <= x< 3.0`
  - `double x = 5*Math.random()+4; // 4.0<=x<9.0`

  In general, to produce a random real number in the range [low,high),

  - `double x = (high-low)*Math.random()+low;`

  Generate a random real value in [7.0,15.0).

  ```
  double x = 8 * Math.random() + 7;
  ```

Random Integers

How do we generate random integers?

```
int x = (int)(100*Math.random());
// random integer 0 to 99 inclusive.
int y = (int)(100*Math.random())+4;
// random integer 4 to 103 inclusive.
int z = (int)(2*Math.random());
// random integer 0 or 1, useful for heads/tails
```

More Examples

```
int x = (int) Math.random()*5;
// x=0
int y = (int)(6*Math.random())-10;
// integer from -10 to -5 inclusive.
double z =3*Math.random()+5;
//random double in [5,8)
```

For vs. While

```
for(int i = 1; i <= 100; i++)
{
      System.out.println(i);
}
```

This for loop executes 100 times. It is a definite loop. It is usually better to use a for loop as a definite loop.

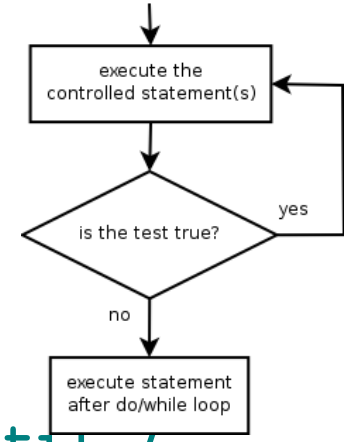
For vs. While

```
int x=3;
while(x == 3)
{
      System.out.println(x);
      x=(int)(4*Math.random()); // x is 0,1,2,or 3
}
```

This while loop executes an unknown number of times. It is a indefinite loop. It is usually better to use a while loop as an indefinite loop.

The `do/while` loop(NOT TESTED)

- **`do/while` loop**: Performs its test at the *end* of each repetition.
  - Guarantees that the loop's `{}` body will run at least once.




```
do {
      statement(s);
} while (test);
```

```
// Example: generate random numbers 0-9 until 7
// is generated.
int rand=(int)(10*Math.random()); ;
do {
     System.out.print(rand+ " ");
     rand=(int)(10*Math.random());
} while (rand != 7);
```

Lab

Create a folder in CS50 called While for this lab.

Write a method `countFactors` that returns the number of factors of an integer. Use a **for loop**.

- `countFactors(24)` returns `8` because 1, 2, 3, 4, 6, 8, 12, and 24 are factors of 24.

Write a method `isPrime` which returns whether or not an integer is prime. This method **must call** `countFactors`.

- Example: `isPrime(27)` returns false and `isPrime(47)` returns true.

Lab

Write a static method named `fourHeads` that repeatedly flips a coin until four heads *in a row* are seen. You should use Math.random() to give an equal chance to a head or a tail appearing. Each time the coin is flipped, what is seen is displayed (H for heads, T for tails). When four heads in a row are flipped a congratulatory message is printed. Here are possible outputs of two calls to fourHeads:

```
T T T H T H H H H
Four heads in a row!

T H T H T T T T T H H T H H H H
Four heads in a row!
```

Lab

Write a static method named `printTwoDigit` that accepts an integer *n* as a parameter and that prints a series of *n* randomly generated numbers. The method should use Math.random() to select numbers in the range of 10 to 19 inclusive where each number is equally likely to be chosen. After displaying each number that was produced, the method should indicate whether the number 13 was ever selected ("we saw a 13!") or not ("no 13 was seen."). You may assume that the value of *n* passed is at least 0.

You should an output similar to below. (see next slide)

Call	`printTwoDigit(4);`	`printTwoDigit(7);`

Output	`next = 12`
	`next = 10`
	`next = 16`
	`next = 11`
	`no 13 was seen.`
		`next = 12`
		`next = 19`
		`next = 12`
		`next = 13`
		`next = 11`
		`next = 16`
		`next = 13`
		`we saw a 13!`